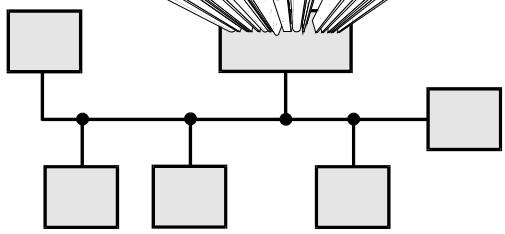
CANalyzer Win



Arbeitshandbuch

Version 2.2

Vector Informatik GmbH, Friolzheimer Str. 6, 70499 Stuttgart
Tel. 0711/139996-0, Fax 0711/139996-30, Email can@vector-informatik.de
Internet http://www.vector-informatik.de

Internationales Qualitätsstandardzertifikat

Die Vector Informatik GmbH, 70499 Stuttgart, Deutschland, hat am 19.08.1998 das ISO 9001-Zertifikat erhalten. Der ISO 9001-Standard ist ein weltweit anerkannter Qualitätsstandard der British Standards Institution.

Zertifikatnummer: 70 100 F 1498 TMS

Vertriebspartner weltweit:

Tochterfirmen:

USA

Vector CANtech, Inc.
Mr. Bruce Emaus
Suite 190
39500 Orchard Hill Place
Novi, Mi 48375
Tel.: +1 248 449-9290
Fax: +1 248 449-9704
Email: bruce@vector-cantech.com
http://www.vector-cantech.com

Japan

Vector Japan Co., Ltd.
Mr. Kentaro Yamauchi
Kyobashi Yayoi Bld. 5F
2-11-6 Kyobashi, Chuo-ku
104-0031 Tokyo
Tel.: +8/-(03)-5524-0902
Fax: +8/-(03)-5524-0905
Email: can@vector-japan.co.jp
http://www.vector-japan.co.jp

Distributoren

USA

HiTOOLS Inc.
Mr. Olaf Pfeiffer
2055 Gateway Place
San Jose, CA 95110
Tel.: +1 408 4513986
Fax: +1 408 4419486
Email: pfeiffer@hitex.com
http://www.hitex.com

Frankreich

NSI S.A.
Mr. Eric Médan
78 bis, route des Creuses
74960 Cran-Gevrier
Tel.: +33 4 50 52 1510
Fax: +33 4 50 52 0885
Email: nsi@nwc.fr

Italien

MICROTASK Embedded S.r.L.
Mr. Marco Cavallaro
P.zza Virgilio, 3
20123 Milano
Tel.: 0039 2 4982051
Fax: 0039 2 4986949
Email: mcavallaro@microtask-embedded.it

Großbritannien

Accutest Limited
Mr. Ken Scott
Suite 2
9 George Street
Glossop
Derbyshire SK13 8AY
Tel.: +44 1457 866613
Fax: +44 1457 856789
Email: info@accutest.co.uk
http://www.accutest.co.uk/index2.html

Schweden

Kvaser AB
Mr. Lars-Berno Fredriksson
P.O. Box 4076
51104 Kinnahult
Tel.: +46 320 15287
Fax: +46 320 15284
Email: lbf@kvaser.se
http://www.kvaser.se

Deutschland

Softing GmbH
Herr Rainer Gallus
Richard-Reitzner Allee 6
85540 Haar
Tel.: +49 89 45656-323
Fax: +49 89 45656-399
Email: rainer.gallus@softing.com
http://www.softing.com

Israel

Dan-EL Technologies Ltd.
Mr. Dan Alon
9, Harugei Hamalchut St.
Tel - Aviv 69714
Tel: 00972-3-6478770
Fax: 00972-3-6478771
Email: danel@danel.co.il
http://www.danel.co.il

Korea

Hanil Mech. Eng. Consultants.
Automatic Control Division
Mr. Dr.-Ing. K. Jun
140-012, 173, 2-KA, Hangang-RO, Yongsan-Ku
Seoul
Phone.: 0082 2 / 709 4804
Fax : 0082 2 / 796 3969

Kopierschutz

Die Software ist nicht kopiergeschützt. Der Lizenznehmer ist berechtigt eine (einzige) Sicherungskopie herzustellen. Er ist verpflichtet, auf dieser Kopie das Urheberrecht der Vector Informatik GmbH zu vermerken. Die Herstellung weiterer Kopien und/oder die Weitergabe der Software an Dritte ist ausdrücklich verboten. Die Bestimmungen des Vector Lizenzvertrags sind einzuhalten.

Die gelieferte Software ist mit einer Seriennummer und mit einem Texteintrag zur Kennzeichnung des Lizenznehmers versehen. Beide Einträge werden beim Start kurzzeitig angezeigt. Zusätzlich können sie über den Menüpunkt *Hilfe|Info* abgerufen werden. Der Lizenznehmer hat mit der notwendigen Sorgfalt darauf zu achten, daß von seiner Version keine unlizensierten Kopien hergestellt werden können. Falls ihm bekannt wird, daß eine Kopie seiner Version von Dritten eingesetzt wird, ist er verpflichtet, dies sofort zu unterbinden oder Vector davon sofort in Kenntnis zu setzen. Andernfalls haftet der Lizenznehmer Vector gegenüber für den dadurch entstandenen Schaden.

Typographische Konventionen

Textformatierung	Bedeutung	Beispiel
Kursiv	Menü-, Befehls- und Dialogfeldnamen	*Datei|Datenbasis zuordnen*
Fett	Beispiele für die CAPL-Syntax	**`on message 0x100 {...}`**
in spitzen Klammen	Tastennamen	<TAB>
Großbuchstaben	Dateinamen	`CAN.INI`
Festbreitenschrift	CAPL-Syntaxbeschreibung	`"DLC" "=" <integer>`

Übersicht

Kapitel 1, **Einführung** stellt den CANalyzer vor und bietet Anfängern auf einer kurzen Tour einen Überblick über die wichtigsten Funktionen.

Kapitel 2, **Arbeiten mit dem CANalyzer** erläutert das Konzept des Meßaufbaus und erklärt die Grundfunktionen, die der CANalyzer zum Senden und Analysieren von Daten bereitstellt.

Kapitel 3, **CAPL - Programmierung** behandelt die CAPL-Programmiersprache, mit denen Sie die Grundfunktionen des CANalyzer praktisch unbegrenzt ergänzen und erweitern können. Die Einsatzmöglichkeiten der Programmiersprache werden an vielen Beispielen verdeutlicht.

Kapitel 4, **Spezielle Themen** bietet Ihnen weitergehende Informationen zum Verständnis der Funktionsweise des CANalyzers und gibt Ihnen zusätzliche Tips und Tricks für den Einsatz des CANalyzers in besonderen Situationen.

Im Kapitel 5, **Anhang** finden Sie Hinweise zur Fehlerbeseitigung und die Beschreibung einiger Zusatzprogramme.

Inhalt

1 Einführung 1-1

1.1 Kurzübersicht 1-1

1.2 CANalyzer-Tour 1-3
1.2.1 Vorbereitungen 1-3
1.2.2 Einrichten des CAN-Busses 1-4
1.2.3 Daten senden 1-6
1.2.4 Auswertefenster 1-9
1.2.5 Arbeiten mit symbolischen Daten 1-11
1.2.6 Analyse von Signalwerten im Datenfenster 1-13
1.2.7 Analyse von Signalverläufen im Grafikfenster 1-15
1.2.8 Einsatz der Datenbasis beim Senden von Botschaften 1-15
1.2.9 Analyse einer Motorraumsimulation 1-16
1.2.10 Aufzeichnen einer Messung 1-17
1.2.11 Auswerten einer Logging-Datei 1-18
1.2.12 Hinweise für das Bearbeiten eigener Aufgabenstellungen 1-19

1.3 Bedienhinweise 1-19
1.3.1 Menübedienung 1-20
1.3.2 Dialoge 1-20
1.3.3 Bedienung des Meßaufbaus 1-22
1.3.4 Das Hilfesystem 1-23

2 Arbeiten mit dem CANalyzer 2-1

2.1 Vorbereiten der Messung 2-1
2.1.1 Programmstart 2-1
2.1.2 Der CANalyzer-Bildschirm 2-2
2.1.3 Datenfluß im Meßaufbau 2-4
2.1.4 Konfiguration des Meßaufbaus 2-6
2.1.5 Arbeiten mit Auswerteblöcken im Meßaufbau 2-7
2.1.6 Messungsstart 2-7
2.1.7 Arbeiten mit Konfigurationen 2-8
2.1.8 Darstellungsformate 2-9

2.2 Die Schnittstelle zur Hardware 2-10
2.2.1 Programmierung der Busparameter 2-11
2.2.2 Akzeptanzfilterung 2-12
2.2.3 Karten- und Treiberoptionen 2-14

2.3 Senden und Empfangen von Daten 2-14
2.3.1 Der Sendezweig 2-14
2.3.2 Die Analysezweige 2-15
2.3.3 Botschaftsattribute 2-15

2.4 Einsatz von Datenbasen 2-17
2.4.1 Erstellen und Modifizieren von Datenbasen 2-18
2.4.2 Zugriff auf Datenbasisinformationen 2-19

2.5 Die Meßfenster 2-20
2.5.1 Trace-Fenster 2-20
2.5.1.1 Einstellungen für das Trace-Fenster 2-21
2.5.1.2 Einstellungen des Trace-Fensters aus der Symbolleiste 2-22

2.5.1.3 Suchen im Trace-Fenster 2-23
2.5.1.3 2-23
2.5.1.4 Trace-Watch-Fenster 2-23
2.5.1.5 Optimierungen im Trace-Fenster 2-23
2.5.2 Datenfenster 2-23
2.5.2.1 Konfiguration der Signale 2-24
2.5.2.2 Anzeigetypen 2-25
2.5.2.3 Aktivitätsanzeige 2-26
2.5.2.4 Peak-Anzeige 2-27
2.5.2.5 Optimierung der Datenanzeige 2-27
2.5.3 Grafikfenster 2-27
2.5.3.1 Auswahl der Signale 2-28
2.5.3.2 Anordnen der Signale 2-29
2.5.3.3 Signal-Layout 2-30
2.5.3.4 Konfiguration der Messung 2-31
2.5.3.5 Meß- und Anzeigefunktionen 2-32
2.5.3.6 Signalmodi 2-32
2.5.3.7 Meßmodi 2-33
2.5.3.8 Anzeigemodi 2-34
2.5.3.9 Layoutfunktionen 2-34
2.5.3.10 Export von Signalen 2-36
2.5.3.11 Symbolleiste des Grafikfensters 2-36
2.5.3.12 Optimierung des Grafikfensters 2-36
2.5.4 Statistik 2-38
2.5.4.1 Direktanzeige im Statistikfenster 2-39
2.5.4.2 Statistikprotokoll 2-40
2.5.5 Busstatistik 2-41
2.5.6 Write-Fenster 2-41
2.5.7 Layout und Fonts der Meßfenster 2-42

2.6 Hot-Spots und einfügbare Funktionsblöcke 2-42
2.6.1 CAPL-Knoten 2-43
2.6.2 Durchlaßfilter 2-45
2.6.3 Sperrfilter 2-45
2.6.4 Replay-Block 2-46
2.6.5 Generatorblock 2-46
2.6.5.1 Konfiguration der Auslösung 2-47
2.6.5.2 Konfiguration der Sendeliste 2-48
2.6.5.3 Eingabe von Signalwerten 2-48
2.6.5.4 Eingabe modusabhängiger Signale 2-49
2.6.5.5 Funktionsgenerator für die Sendeliste 2-50
2.6.6 Unterbrechung 2-52

2.7 Aufzeichnen und Auswerten von Meßdateien 2-52
2.7.1 Logging-Trigger 2-52
2.7.1.1 Triggermodi 2-53
2.7.1.2 Trigger-Ereignisse 2-55
2.7.1.3 Zeitfenster 2-56
2.7.1.4 Konfigurieren des Logging-Puffers 2-56
2.7.2 Logging-Dateien 2-57
2.7.3 Logging-Dateiformate 2-58
2.7.4 Datenanalyse im Offline-Modus 2-59
2.7.4.1 Ablaufsteuerung im Offline-Modus 2-61
2.7.4.2 Konfiguration von Online- und Offline-Modus 2-62
2.7.5 Trigger- und Suchbedingungen 2-62
2.7.5.1 Bedingungsprimitive 2-63
2.7.5.2 Eingabe von ID und Datenbytes 2-63
2.7.5.3 Symbolische Triggerung 2-64

3 CAPL - Programmierung 3-1

3.1 Übersicht 3-1
3.1.1 Einsatzmöglichkeiten von CAPL-Programmen 3-1
3.1.2 Einbinden von CAPL-Programmen 3-2
3.1.3 Einsatz der symbolischen Datenbasis in CAPL 3-3

3.2 Einführung in CAPL 3-4
3.2.1 CAPL Variablen 3-4
3.2.2 Deklaration von Botschaften 3-6
3.2.3 Ereignisprozeduren 3-8
3.2.4 Ausdrücke in CAPL 3-14
3.2.5 Das Schlüsselwort this 3-15
3.2.6 Interpretation von Signalwerten 3-15
3.2.7 Zugriff auf Attribute in CAPL 3-16
3.2.8 Arbeiten mit symbolischen Signalwerten 3-17
3.2.9 Arbeiten mit mehreren Datenbasen 3-18
3.2.10 Funktionen in CAPL 3-19

3.3 Erstellen eines CAPL-Programms 3-20

3.4 Browser zum Erstellen und Compilieren der CAPL-Programme 3-21
3.4.1 Öffnen des Browsers 3-22
3.4.2 Browser-Fenster 3-23
3.4.3 Compilieren von CAPL-Programmen 3-23
3.4.4 Suchen von Laufzeitfehlern 3-24
3.4.5 Zugriff auf die Datenbasis 3-24
3.4.6 Importieren und Exportieren von ASCII-Dateien 3-25
3.4.7 Browser-Optionen 3-25

3.5 CAPL Sprachbeschreibung 3-26
3.5.1 Erweiterte Backus-Naur-Form 3-26
3.5.2 Programmaufbau 3-26
3.5.3 Deklaration von Variablen 3-26
3.5.4 Ereignisgesteuerte Prozeduren 3-30
3.5.5 CAPL-Unterprogramme 3-39

3.6 Spezielle Themen in CAPL 3-41
3.6.1 Einbinden einer Windows DLL in CAPL 3-41
3.6.2 Behandlung von Laufzeitfehlern 3-43

3.7 Liste der eingebauten Funktionen 3-43
3.7.1 Zugriff auf CAN und andere Hardware 3-43
3.7.2 Benutzerinteraktion 3-48
3.7.3 Logging-Funktionen 3-50
3.7.4 Standard / Extended Identifier 3-51
3.7.5 Ablaufkontrolle 3-52
3.7.6 Zeitverwaltung 3-53
3.7.7 Zugriff auf CANoe-Umgebungsvariablen 3-56
3.7.8 Intel/Motorola-Formate 3-58
3.7.9 Trigonometrie und mathematische Funktionen 3-59
3.7.10 Dateifunktionen 3-60
3.7.11 Sequentieller Dateizugriff 3-63
3.7.12 String-Funktionen 3-66
3.7.13 Sprachunterstützung und Debugging 3-67

3.8 Beispiele 3-68
3.8.1 Senden von Botschaften 3-68

3.8.2 Gateway-Anwendung....3-72
3.8.3 CAPL-gesteuerte Triggerung....3-73
3.8.4 Verwendung der Symboldatenbasis....3-74
3.8.5 Erstellen eines Netzknotenmodells mit CAPL in CANoe....3-76

4 Spezielle Themen....4-1

4.1 Systembeschreibung....4-1
4.1.1 Übersicht über die Programme....4-1
4.1.2 CANalyzer Architektur....4-1

4.2 Der CANalyzer im Last- und Überlastbetrieb....4-2
4.2.1 Verhalten in Lastsituationen....4-2
4.2.2 Verhalten bei Datenverlust....4-3
4.2.3 Füllstandsanzeige....4-3
4.2.4 Konfigurationsmöglichkeiten bei hoher Buslast....4-4

4.3 Arbeiten mit Datenbasen....4-5
4.3.1 Zuordnen von Datenbasen....4-6
4.3.2 Verwendung mehrerer Datenbasen....4-7
4.3.3 Auflösung von Mehrdeutigkeiten....4-7
4.3.4 Überprüfung der Konsistenz symbolischer Daten....4-8

4.4 Die DDE-Schnittstelle....4-8
4.4.1 DDE-Dienste....4-9
4.4.2 Das DDE Thema *System*....4-9
4.4.3 Die DDE Transaktion *Execute*....4-10
4.4.4 Laden von Konfigurationen mit dem Explorer....4-11

5 Anhang....5-1

5.1 Besonderheiten der Demo-Version....5-1

5.2 Verwendete Dateinamen....5-2

5.3 Trouble shooting....5-3

5.4 Meldungen im Write-Fenster....5-4

5.5 Liste der Fehlermeldungen zur CAN-Schnittstelle....5-8

5.6 Zusatzprogramme....5-12
5.6.1 ASC2MDF.EXE....5-12
5.6.2 CANCONV.EXE....5-13
5.6.3 LOG2TXT.EXE....5-13

6 Index....6-1

1 Einführung

1.1 Kurzübersicht

Der CANalyzer ist ein universelles Entwicklungswerkzeug für CAN-Bussysteme, mit dessen Hilfe der Datenverkehr auf der Busleitung beobachtet, analysiert und ergänzt werden kann. Durch leistungsfähige Funktionen und durch freie Programmierbarkeit werden alle Belange von den ersten Gehversuchen mit CAN bis zur gezielten Fehlersuche bei komplexen Problemstellungen abgedeckt.

Sie arbeiten mit dem CANalyzer wahlweise auf Byteebene im busnahen Rohdatenformat oder aber auf Applikationsebene mit logisch/physikalischer Datendarstellung. Zur Umsetzung der Rohdaten wird eine CAN-Datenbasis verwendet, die in der Kfz-Industrie ein De-facto-Standard ist. Zur Erstellung und Änderung der Datenbasis steht Ihnen das komfortable Verwaltungsprogramm CANdb zur Verfügung, das zum Lieferumfang des CANalyzers gehört.

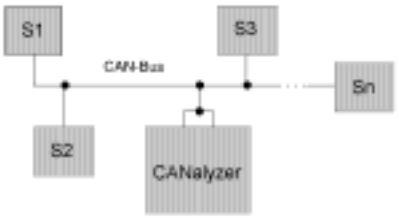
Bereits die eingebauten Grundfunktionen, zu deren Benutzung keine Programmierkenntnisse erforderlich sind, bieten eine Fülle von Einsatzmöglichkeiten. Dazu gehören u.a. die Auflistung des Busdatenverkehrs (Tracing), die Anzeige von Datensegmenten bestimmter Botschaften, die grafische Darstellung und Analyse von zeitlichen Signalverläufen, das Senden vordefinierter Botschaften, die statistische Auswertung von Botschaften und Busstörungen sowie die Aufzeichnung von Botschaften zur Wiedergabe oder zur Offline-Auswertung.

Darüber hinaus können Sie die Funktionalität des CANalyzers durch die freie Programmierbarkeit beliebig erweitern. An jeder Stelle im Datenflußplan des CANalyzers lassen sich Programmblöcke einfügen. Als Programmiersprache dient die anwendungsorientierte, C-ähnliche Sprache CAPL (CAN Access Programming Language). Ein spezielles Ereignisprozedurkonzept und CAN-angepaßte Sprachmittel erlauben die schnelle Entwicklung problemspezifischer Lösungen. Eine interaktive Entwicklungsumgebung - der CAPL-Browser - zur einfachen Erstellung, Modifikation und Compilierung von CAPL-Programmen ist im CANalyzer enthalten.

Durch die Programmierbarkeit ergeben sich vielfältige Einsatzmöglichkeiten:

Emulation einer Busstation:

Dazu kann der CANalyzer die für den Datenverkehr relevanten Funktionen der Busstation nachbilden, z.B. Senden von Botschaften als Folge bestimmter Ereignisse.

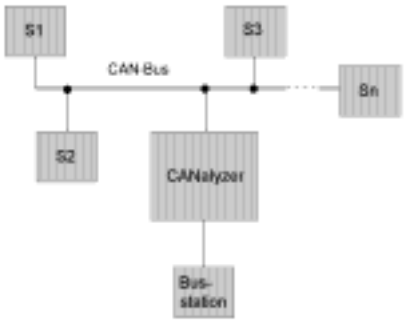


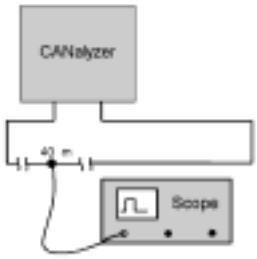
Emulation der Systemumgebung für den Test einer Busstation:

Bei der Entwicklung von Geräten mit Busanschluß entsteht das Problem, daß für Tests die restlichen Busteilnehmer noch nicht verfügbar sind. Zur Emulation der Systemumgebung kann mit Hilfe des CANalyzers der Datenverkehr aller restlichen Stationen nachgebildet werden.

Verbindungselement zwischen zwei Bussen:

Damit können Daten zwischen verschiedenen, evtl. unterschiedlich schnellen CAN-Bussen ausgetauscht werden. Eine andere Zielanwendung ist die Zwischenschaltung des CANalyzers zwischen eine zu testende Busstation und den eigentlichen Bus zur Beobachtung und testweisen Manipulation des Datentransfers.





Testgenerator zur Untersuchung des Physical Layers:

Es ist möglich, die Botschaften gezielt auf die beiden Busanschlüsse des CANalyzers zu verteilen und eine echte (lange) Busleitung anzuschließen. Dadurch lassen sich auch sehr einfach Versuche mit der Arbitrierung oder mit Leitungsreflexionen durchführen.

Die Bedienung und Konfigurierung des CANalyzers erfolgt anhand des Datenflußplans im Meßaufbaufenster. Weitere Informationen hierzu finden Sie in Abschnitt 2.1.4.

1.2 CANalyzer-Tour

Falls Sie den CANalyzer zum ersten Mal in Betrieb nehmen und Funktionsweise und Bedienung für Sie noch völlig neu ist, hilft Ihnen die folgende Tour, in ca. zwei Stunden mit dem Bedienkonzept und den wichtigsten Features vertraut zu werden.

Sie werden für diese Tour zunächst einen sehr einfachen CAN-Bus aufbauen, bei dem der CANalyzer sowohl die Rolle des Senders als auch die des Empfängers übernimmt. Im ersten Schritt wird der CANalyzer als Datenquelle, d.h. als Sendestation, konfiguriert. Sie werden dann die Analysemöglichkeiten des CANalyzers kennenlernen, indem Sie die erzeugten Daten anschließend in den Meßfenstern des CANalyzers untersuchen.

Auch in komplexeren realen Systemen übernimmt der CANalyzer typischerweise beide Aufgaben. Sie können ihn als Datenquelle einsetzten, um Daten an andere Steuergeräte zu senden, Sie können ihn aber gleichzeitig verwenden, um den Datenverkehr am CAN-Bus zu beobachten, aufzuzeichnen und auszuwerten.

1.2.1 Vorbereitungen

Zum Start des CANalyzers rufen Sie `CANW.EXE` (16-bit Version) bzw. `CANW32.EXE` (32-bit Version) durch Doppelklicken auf das entsprechende Symbol in der CANalyzer-Programmgruppe auf.

Als Windows-Programm verfügt der CANalyzer über verschiedene Auswertefenster (Trace-Fenster, Datenfenster, Grafikfenster, Statistik- und Busstatistik-Fenster) sowie ein Meßaufbaufenster, das Ihnen den Datenfluß im CANalyzer anzeigt und über das Sie den CANalyzer gleichzeitig konfigurieren können.

Sie erreichen alle Fenster des CANalyzers über das *Ansicht*-Menü in der Hauptmenüzeile.

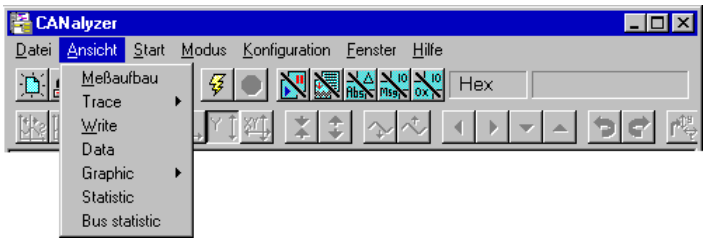


Abbildung 1: *Ansicht*-Menü Hauptmenüzeile

Das Datenflußdiagramm des CANalyzer-Meßaufbaus enthält links die Datenquelle - symbolisiert durch das Symbol einer PC-Einsteckkarte - und rechts verschiedene Auswerteblöcke als Datensenken. Die Daten fließen also von rechts nach links. Zur Veranschaulichung des Datenflusses sind zwischen den einzelnen Elementen Verbindungsleitungen und Verzweigungen eingezeichnet.

Die Informationen die in jedem Auswerteblock eintreffen, werden im zugehörigen Auswertefenster dargestellt. So stellt z.B. das Trace-Fenster alle Informationen dar, die im Trace-Block ankommen, während das Grafikfenster Ihnen die Informationen anzeigt, die im Grafikblock eintreffen.

Einzige Ausnahme ist der Logging-Block, dem kein Fenster, sondern eine Datei zugeordnet ist, in der die am Block eintreffenden Daten aufgezeichnet werden.

Auf der linken Seite des Meßaufbaus zweigt hinter dem Kartensymbol der Sendezweig des CANalyzers ab. Von hier aus können Sie Daten auf den Bus senden. Der Datenfluß verläuft im Sendezweig stets von oben nach unten.

Im Datenflußdiagramm erkennen Sie ferner kleine schwarze Quadrate: ⊣■⊢. An diesen Einfügepunkten (Hot-Spots) können Sie weitere Funktionsblöcke zur Manipulation des Datenflusses (Filter, Replay, Generatorblock, CAPL-Programmblock mit benutzerdefinierbarer Funktionen) einfügen.

Stellen Sie sicher, daß Sie diese Tour mit einen neuen Konfiguration beginnen, indem Sie den Menüpunkt *Datei|Neue Konfiguration* auswählen.

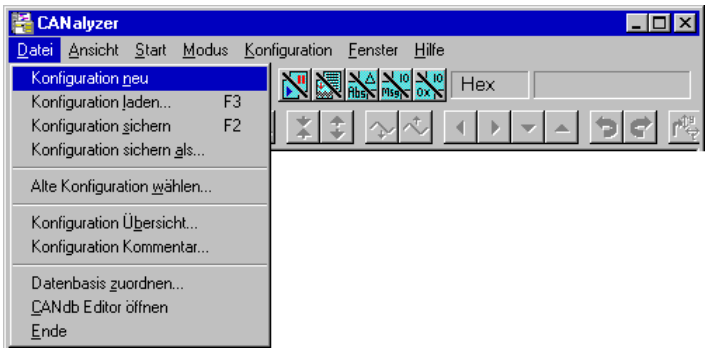


Abbildung 2: Menüpunkt *Datei|Neue Konfiguration*

1.2.2 Einrichten des CAN-Busses

Zur Inbetriebnahme des CANalyzers bietet es sich an, einen von vorhandenen CAN-Bussystemen unabhängigen Versuchsaufbau mit lediglich zwei Netzknoten zu verwenden. Als Netzknoten dienen dabei die beiden CAN-Controller der mitgelieferten PC-Karte.

Verbinden Sie zunächst die beiden D-Sub-9-Stecker Ihrer CAN-Karte miteinander. Ein Verbindungskabel mit zwei Busabschlußwiderständen von je 120Ω ist im Lieferumfang von CANalyzern mit High-Speed-Busankopplung enthalten. Bei einer Low-Speed-Ankopplung benötigen Sie lediglich ein 3-adriges Kabel, um die den Busleitungen CAN-High, CAN-Low und Masse zugeordneten Pins beider Controller miteinander zu verbinden.

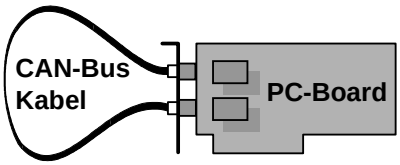


Abbildung 3: PC-Karte mit Verbindungskabel

Der CAN-Bus, den Sie während dieser Tour verwenden, besteht also aus einem kurzen 2- bzw. 3 adrigen Kabel, das die beiden CAN-Controller der CAN-Karte miteinander verbindet. Diese minimale Konfiguration ist erforderlich, da das CAN-Protokoll neben einem Sender mindestens einen Empfänger erfordert, der korrekt empfangene Botschaften mit einem Acknowledge quittiert.

Offen geblieben ist bis jetzt die Festlegung der Busparameter (Übertragungsgeschwindigkeit, Abtastzeitpunkt, ...), die an jeden der beiden beteiligten Controller eingestellt werden müssen. Holen Sie zu diesem Zweck den Meßaufbau über das *Ansicht*-Menü in den Vordergrund, und klicken Sie mit der rechten Maustaste auf das PC-Kartensymbol links in diesem Fenster.

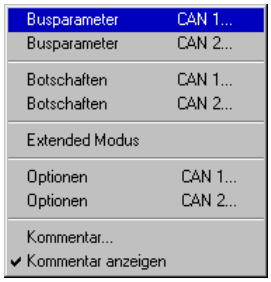


Abbildung 4: Kontextmenü des PC-Kartensymbols

Wählen Sie im Kontextmenü die Busparameter zunächst für den ersten Controller CAN 1 und stellen Sie im Konfigurationsdialog zunächst die Baudrate ein: Mit der Schaltfläche *Baudratenauswahl* gelangen Sie in die Baudratendirektauswahl, wo Sie den Wert 100 kBaud eintragen.

Abbildung 5: Konfiguration Busparameter und Baudratendirektauswahl

Dieser ist sowohl für High-Speed als auch für Low-Speed-Busse sinnvoll. Nach Betätigen der Schaltfläche *Aktualisieren* schlägt Ihnen der CANalyzer Default-Werte für die Controller-Register vor, die Sie mit *OK* übernehmen. Damit haben Sie neben der Übertragungsgeschwindigkeit von 100 kBaud implizit auch die anderen Controller-Parameter (Abtastzeitpunkt, BTL-Zyklen und Synchronisationssprungweite) festgelegt. Damit das Gesamtsystem funktionieren kann, müssen diese Werte genau so für den zweiten Controller CAN 2 übernommen werden. Der CANalyzer fragt Sie beim Verlassen des Dialogs, ob die Parameter übernommen werden sollen. Antworten Sie mit *JA*.

1.2.3 Daten senden

Da Ihr aktueller Versuchsaufbau noch keine Datenquellen enthält, besteht Ihre erste Aufgabe darin, im CANalyzer eine Datenquelle einzurichten, die Informationen zyklisch auf den Bus legt.

Aufgabe 1: Konfigurieren Sie den CANalyzer so, daß nach Messungsstart alle 100 Millisekunden eine CAN-Botschaft mit dem Identifier 64 (hex) auf den Bus gelegt wird. Die Botschaft soll dabei genau vier Datenbytes mit den Werten D8 (hex), D6 (hex), 37 (hex) und 0 enthalten.

Sie lösen diese Aufgabe, indem Sie im Sendezweig des CANalyzers einen Generatorblock einfügen, der die zu sendende Botschaft generiert. Klicken Sie dazu mit der rechten Maus auf den Hot-Spot im Sendezweig direkt über dem Block mit der Aufschrift *Senden* und fügen Sie über das Kontextmenü des Hot-Spots einen Generatorblock in den Sendezweig ein.

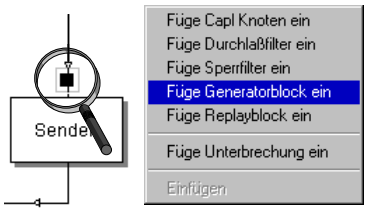


Abbildung 6: Hot-Spot im Sendezweig mit Kontextmenü

Dieser erscheint danach als quadratischer Block mit der Aufschrift *G*. Sie konfigurieren ihn wiederum über sein Kontextmenü, das Sie über die rechte Maustaste erhalten.

Füllen Sie zunächst die Sendeliste aus. Als Identifier tragen Sie 64 ein. (Überprüfen Sie über die *Optionen*-Schaltfläche, ob das Zahlenformat *Hex* eingestellt ist.) Als Datenlängenangabe tragen Sie danach im Feld DLC den Wert 4 ein. Die Werte der Datenbytes legen Sie schließlich in den folgenden vier Datenfeldern fest, indem Sie dort die Werte D8, D6, 37 und 0 eintragen.

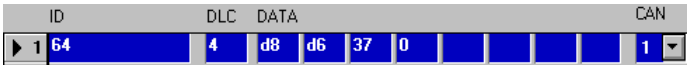


Abbildung 7: Sendeliste Generatorblock

Verlassen Sie die Sendeliste mit *OK*, um die Werte in die Konfiguration zu übernehmen. Im Kontextmenü des Generatorblocks müssen Sie nun noch die Auslösung für den Sendevorgang konfigurieren. In der zweiten Zeile kreuzen Sie *Mit Periode* an, wobei Sie anschließend im Eingabefeld rechts daneben den Wert 100 eintragen.

Abbildung 8: Auslösung Generatorblock

Mit *OK* werden auch diese Werte in die Konfiguration übernommen.

Bevor Sie die Messung starten, sollten Sie Ihre bis hierhin vorbereitete Konfiguration mit dem Menüeintrag *Datei|Konfiguration sichern* abspeichern. Sie können die Konfiguration dann jederzeit wieder laden und Ihre Arbeit genau an diesem Punkt fortführen.

Starten Sie die Messung, indem Sie den Startknopf in der Symbolleiste drücken. Der CANalyzer beginnt unmittelbar mit dem periodischen Senden der Botschaft, die Sie im Generatorblock konfiguriert haben. Sie erkennen dies im Trace-Fenster, das nach Messungsstart automatisch in den Vordergrund springt und nun rechts unten im CANalyzer zu sehen ist: In der ersten Zeile sehen Sie die Botschaft, die vom Generatorblock gesendet wird, wobei in der ersten Spalte der Sendezeitpunkt relativ zum Messungsstart angezeigt wird.

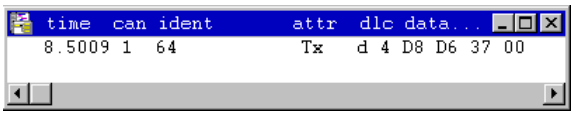


Abbildung 9: Trace-Fenster

Die nächste Spalte zeigt Ihnen an, über welchen der beiden CAN-Controller gesendet wird. Dieser Wert (CAN 1) stimmt mit dem Default-Wert überein, der in der Sendeliste des Generatorblocks für zu versendende Botschaften vergeben wird.

Über den Bus wird diese Botschaft danach auch vom zweiten CAN-Controller empfangen. Die Frage liegt nahe, warum sie dann aber nicht im Trace-Fenster angezeigt wird? Eine Antwort finden im Konfigurationsdialog des Akzeptanzfilters für den zweiten Controller. Sie öffnen diesen Dialog wiederum über das Kontextmenü des PC-Kartensymbols im Meßaufbau unter dem Eintrag *Botschaften CAN 2*.

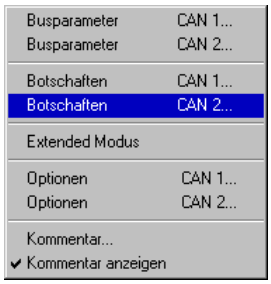


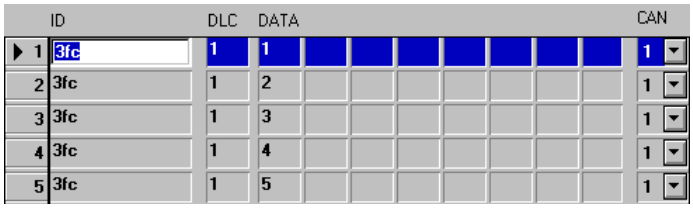
Abbildung 10: Kontextmenü des PC-Kartensymbols

Die Akzeptanzfilterungeinstellungen unterstützen eine hardwareseitige Filterung der Botschaften. Die Default-Einstellungen sperren den Botschaftsempfang weitgehend. Sie öffnen das Filter, indem Sie in die oberen Zeile mit dem Wert *X* füllen. Nach erneutem Messungsstart können Sie nun auch erkennen, daß die über Kanal 1 gesendete Botschaft (Sendeattribut Tx [= Transmit] im Trace-Fenster) vom zweiten Controller empfangen wird (Empfangsattribut Rx [= Receive] im Trace-Fenster).

Wir werden nun die Aufgabe erweitern und zusätzlich eine Botschaft mit veränderlichen Daten senden:

Aufgabe 2: Erweitern Sie die Konfiguration aus der letzten Aufgabe so, daß zusätzlich alle 200 Millisekunden eine Botschaft mit dem Identifier 3FC (hex) gesendet wird, der Wert des ersten Datenbytes soll dabei zyklisch die Werte von 1 bis 5 annehmen.

Sie lösen diese Aufgabe, indem Sie in den Sendezweig einen weiteren Generatorblock einfügen. Ob Sie ihn vor oder hinter dem ersten Generatorblock einfügen, spielt für diese Aufgabenstellung keine Rolle. Als Wert für die periodische Auslösung wählen Sie *200 ms*. Die Sendeliste sollte wie folgt aussehen:



	ID	DLC	DATA								CAN
1	3fc	1	1								1
2	3fc	1	2								1
3	3fc	1	3								1
4	3fc	1	4								1
5	3fc	1	5								1

Abbildung 11: Sendeliste Generatorblock

Vergessen Sie nicht, die Messung zu stoppen, bevor Sie den Meßaufbau umkonfigurieren. Während einer laufenden Messung sind Änderungen der Meßaufbaukonfiguration nicht möglich. Die Menüpunkte der entsprechenden Kontextmenüs erscheinen gegraut.

Neben dem Generatorblock stellt Ihnen der CANalyzer zwei weitere Blöcke als Datenquellen-Typen bereit. Mit dem Replay-Block können Sie Daten, die mit der Logging-Funktion des CANalyzers aufgezeichnet wurden, wieder auf den Bus abspielen. Der Programmblock bietet Ihnen die Möglichkeit, mit der Programmiersprache CAPL (vgl. Kapitel 3) eigene u.U. auch recht komplexe Sendefunktionalitäten zu programmieren und in den CANalyzer einzubinden.

1.2.4 Auswertefenster

Um die Daten zu analysieren, die nun von den Generatorblöcken im Sendezweig erzeugt werden, setzen Sie die Auswertefenster ein.

Das Trace-Fenster haben Sie bereits kennen gelernt. Hier werden die Daten, die in den Trace-Block des Meßaufbaus gelangen, in einem busnahen Format als CAN-Botschaften dargestellt. Neben dem Zeitstempel gehört dazu die Nummer des CAN-Controllers, der Identifier, ein Attribut zur Unterscheidung von gesendeten bzw. empfangenen Botschaften sowie die Datenbytes der CAN-Botschaft. Sie können das Trace-Fenster, wie alle Analysefenster über das Kontextmenü konfigurieren, das Sie durch Klicken mit der rechten Maustaste auf das Fenster bzw. auf den zugehörigen Block erhalten.

Zur Konfiguration des Trace-Fensters stehen Ihnen ferner die vier rechten Schaltflächen der Symbolleiste zur Verfügung. Mit können Sie beispielsweise vom "feststehenden" Modus in den Scroll-Modus umschalten, in dem jede im Trace-Block eintreffende Botschaft in einen neue Zeile geschrieben wird.

Mit schalten Sie von absoluter auf relative Zeitdarstellung um. In der relativen Zeitdarstellung wird in der ersten Spalte die Zeitdifferenz zwischen zwei aufeinanderfolgenden Botschaften („Sendeabstand“) dargestellt. In diesem Darstellungsformat finden Sie natürlich auch leicht den Sendeabstand wieder, den Sie vorhin im Generatorblock eingetragen hatten: 100 Millisekunden.

Busnahe Informationen bietet Ihnen auch das Statistikfenster. Hier können Sie die Sendehäufigkeiten der Botschaften aufgeschlüsselt nach Identifiern beobachten. Falls Sie den Sendezweig so konfiguriert haben, wie in den beiden letzten Aufgaben vorgegeben, dann sollten Sie im Statistikfenster nach Messungsstart zwei vertikale

Linien erkennen, welche die Sendehäufigkeiten der beiden generierten Botschaften 64(hex) und 3FC(hex) darstellen.

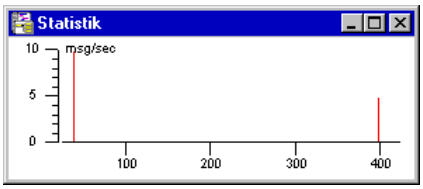


Abbildung 12: Statistikfenster

Für den Identifier 64 werden 10 Botschaften pro Sekunde registriert, für den Identifier 3FC sind es gerade halb so viele. Dieses Ergebnis entspricht den in den Generatorblöcken eingestellten Periodendauern von 100 bzw. 200 Millisekunden.

Falls Ihnen die grafische Anzeige des Fensters zu ungenau ist, bietet Ihnen der Statistikblock ein Statistikprotokoll, das Ihnen genauere Aufschlüsse über den Sendeabstand jeder Botschaft liefert. Stoppen Sie die Messung und aktivieren Sie das Statistikprotokoll im Kontextmenü des Statistikfensters (*Statistikprotokoll anlegen*).

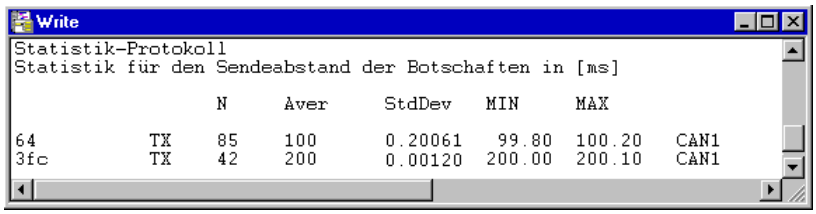

Write
Statistik-Protokoll
Statistik für den Sendeabstand der Botschaften in [ms]
N Aver StdDev MIN MAX
64 TX 85 100 0.20061 99.80 100.20 CAN1
3fc TX 42 200 0.00120 200.00 200.10 CAN1

Abbildung 13: Statistikprotokoll

Wenn Sie nun die Messung erneut starten, werden im Hintergrund statistische Informationen gesammelt, die Sie nach erneutem Messungsstopp über den Kontextmenüpunkt *Statistikprotokoll anzeigen* ins Write-Fenster ausgeben können.

Das Statistikprotokoll bietet neben der Gesamtzahl der Botschaften für jeden Identifier den Durchschnittswert, die Standardabweichung sowie Minimum und Maximum für den registrierten Sendeabstand.

Als weiteres busnahes Fenster bietet Ihnen das Busstatistikfenster eine Gesamtübersicht über den Busdatenverkehr. Hier werden die Gesamthäufigkeiten von Daten-, Remote-, Error- und Overload-Frames, die Busauslastung sowie der Zustandes der CAN-Controller angezeigt. Da in unserem Fall eine Botschaft alle 100 ms und die zweite Botschaft alle 200 ms gesendet wird, liegt die Gesamthäufigkeit aller Botschaften bei 15 Frames pro Sekunde. Bei einer durchschnittlichen Datenlänge von etwa 70 Bit pro Frame sollten in der Sekunde ca. 15 * 10 ≈ 1000 Bit auf den Bus gelangen. Bei einer Baudrate von 100 kBit/sec liegt folglich die Buslast in unserem Beispiel in der Größenordnung von einem Prozent.

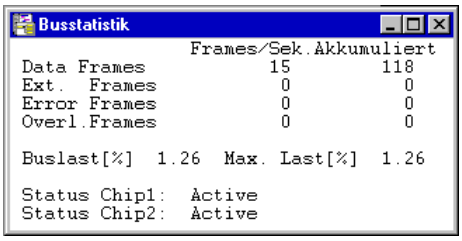


Abbildung 14: Busstatistikfenster

1.2.5 Arbeiten mit symbolischen Daten

Bevor wir auf die verbleibenden Fenster näher eingehen, werfen wir einen Blick auf die Möglichkeiten, die der CANalyzer zur symbolischen Beschreibung von Daten bietet. Von Interesse sind bei der Analyse von CAN-Systemen neben busnahen Informationen, wie Botschaften, Error-Frames und Botschaftshäufigkeiten vor allen die Nutzinformationen, d.h. Signale, wie Drehzahl, Temperatur oder Motorlast, die von einzelnen Steuergeräten bereitgestellt und mit Hilfe von CAN-Botschaften über den Bus gesendet werden.

Um diese Informationen symbolisch zu beschreiben, stellt Ihnen der CANalyzer das Datenbasis-Format CANdb einschließlich eines Datenbasis-Editors zur Verfügung, mit dem Sie CANdb-Datenbasen lesen, erstellen und modifizieren können. Detaillierte Informationen zur CAN-Datenbasis finden Sie im CANdb-Handbuch, das im Lieferumfang des CANalyzers enthalten ist.

Wir wollen an dieser Stelle der aktuellen CANalyzer-Konfiguration eine vorgegebene Datenbasis zuordnen, mit der sich die Datenbytes der Generatorblöcke im Sendezweig interpretieren lassen. Die Datenbasis `MOTBUS.DBC` befindet sich im Demo-Verzeichnis `DEMO_CL` des CANalyzers, das parallel zum Systemverzeichnis `EXEC` bzw. `EXEC32` angelegt wurde, sofern Sie die Demo-Konfigurationen mit installiert haben. Orden Sie diese Datenbasis der aktuellen CANalyzer-Konfiguration über den Menüpunkt *Datei|Datenbasis zuordnen* zu. Sie können nun die Datenbasis über die Schaltfläche der Symbolleiste öffnen.

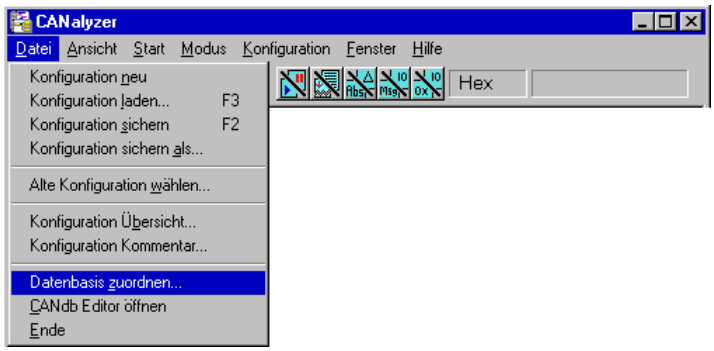


Abbildung 15: Menüpunkt *Datei|Datenbasis zuordnen*

Im Datenbasis-Editor erkennen Sie eine Liste mit vier Botschaften, deren Identifier links und deren symbolische Namen rechts daneben dargestellt werden. Schalten Sie das Zahlenformat im Menüpunkt *Optionen|Allgemeine Einstellungen* zunächst von dezimal auf hexadezimal um. Die symbolischen Namen der Botschaften deuten bereits darauf hin, daß es sich bei dem betrachteten System um die Beschreibung der Kommunikation eines rundimentären Motorraumsystems handelt.

Doppelklicken Sie auf die Botschaft *EngineData* mit dem Identifier 64 (hex), um sich die Botschafts- und Signalparameter dieser Botschaft genauer anzusehen. Die Botschaft trägt 3 Signale: Die Drehzahl (*EngSpeed*), die Temperatur (*EngTemp*) und einen Leerlaufschalter (*IdleRunning*).

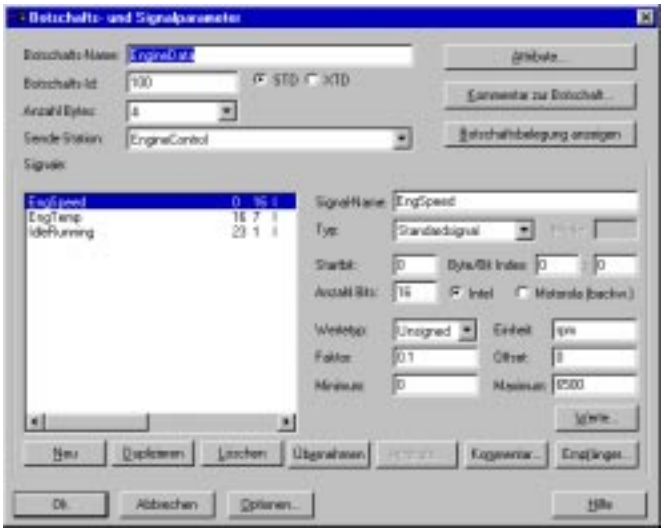


Abbildung 16: Datenbasis

Klicken Sie auf jedes der drei Signale in der Signalliste, um die jeweiligen Parameter im rechten Teil des Dialogs zu untersuchen. Bei der Drehzahl handelt es sich um ein 16-Bit-Signal, das mit dem Faktor 0.1 multipliziert werden muß, um den physikalischen Wert in Umdrehungen pro Minute zu erhalten. Die Temperatur ist in den nächsten 7 Bit der Botschaft kodiert. Um den physikalischen Wert in Grad Celsius zu erhalten, muß der Bitwert mit 2 multipliziert und vom Ergebnis dann 50 subtrahiert werden. Der Leerlaufschalter im letzten Bit des dritten Datenbytes ist schließlich ein binäres Signal (ein Bit), das die Werte 0 und 1 annehmen kann.

Mit Hilfe dieser symbolischen Information lassen sich nun die Dateninhalte von Botschaften im CANalyzer interpretieren. Beachten Sie, das dies natürlich nur dann Sinn macht, wenn die Information der Datenbasis das System beschreibt, das Sie gerade beobachten. Sie können dem CANalyzer natürlich auch willkürlich eine andere Datenbasis zuordnen. Er wird dann den beobachteten CAN-Datenverkehr gemäß den Informationen in dieser Datenbasis interpretieren, auch wenn dies keinen Sinn macht. Sie selber sind verantwortlich, daß die dem CANalyzer zugeordnete Datenbasis und das reale CAN-Netzwerk zusammenpassen.

Die Botschaften, die Sie in den beiden Generatorblöcken erzeugen, lassen sich mit der Datenbasis `MOTBUS.DBC` interpretieren. Beachten Sie, daß die Botschaft, die Sie in der ersten Aufgabe generiert haben, den Identifier 64 (hex) besitzt. Dieser stimmt mit dem Identifier der Botschaft *EngineData* überein, die wir eben im Datenbasis-Editor untersucht haben. Wenn Sie nun also die Messung starten, so können Sie durch Betätigen der Schaltfläche den CANalyzer in den symbolischen Modus umschalten. Im Trace-Fenster sehen Sie jetzt statt des Identifiers den symbolischen Botschaftsnamen.

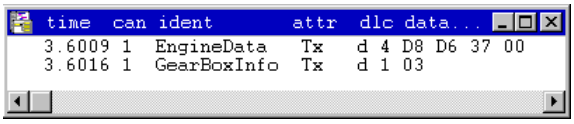


Abbildung 17: Trace-Fenster

1.2.6 Analyse von Signalwerten im Datenfenster

Neben den symbolischen Botschaftsnamen lassen sich mit der zugeordneten Datenbasis auch Signalwerte analysieren. Zur Untersuchung der momentanen Signalwerte dient das Datenfenster.

Damit ist klar, warum das Datenfenster in einer neuen Konfiguration zunächst leer ist: Die anzuzeigenden Signalwerte hängen allein von der Information aus der Datenbasis ab. Welche Signalwerte angezeigt werden sollen, müssen Sie als Anwender entscheiden.

Aufgabe 3: Konfigurieren Sie das Datenfenster so, daß die Signalwerte der im Sendezweig erzeugten Botschaft *EngineData* (ID 64 hex) angezeigt werden.

Um diese Aufgabe zu lösen, öffnen Sie zunächst das Kontextmenü des Datenfensters und starten dann den Konfigurationsdialog. Die Signalliste darin ist zunächst noch leer. Mit der Schaltfläche *Neues Signal* starten Sie den Signal-Explorer, der es Ihnen ermöglicht, ein Signal aus der Datenbasis auszuwählen. Die

Objekthierarchie auf der linken Seite des Dialoges ermöglicht es Ihnen gezielt nach einem Signal zu suchen. Auf der rechten Seite stehen die Signale des ausgewählten Objektes.

Um das Datenfenster zu konfigurieren, wählen Sie also zunächst in der Liste aller Botschaften *EngineData* aus.

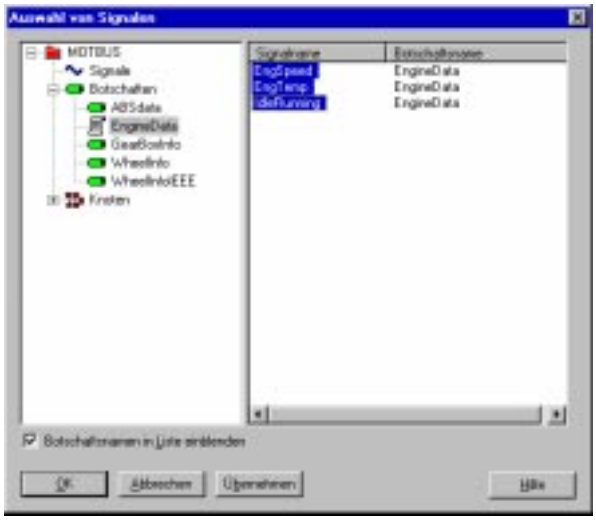


Abbildung 18: Auswahl von Signalen mit dem Signal-Explorer

Danach selektieren und übernehmen Sie alle Signale dieser Botschaft aus der rechten Dialogliste.

Wenn Sie den Konfigurationsdialog des Datenfensters schließen, sehen Sie, daß die Signalnamen nun im Fenster eingetragen sind. Nach Messungsstart beginnt der Generatorblock die Botschaft *EngineData* zyklisch mit den Datenbytes D8, D6, 37 und 0 auf den Bus zu legen. Nach der Botschaftsbeschreibung in der Datenbasis, interpretiert der CANalyzer diese Bytewerte nun als Drehzahl, Temperatur und Leerlaufschalter und stellt die entsprechenden Signalwerte in ihren physikalischen Einheiten im Datenfenster dar.

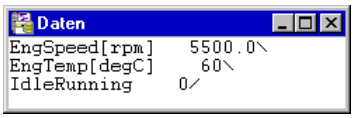


Abbildung 19: Datenfenster

Die Drehzahl wird also mit Hilfe der Umrechnungsformel der Datenbasis in Umdrehungen pro Minute angezeigt, während Sie die Temperatur in Grad Celsius sehen. Die Werte aller drei Signale bleiben zeitlich konstant, da vom Generatorblock stets die Botschaft mit den gleichen Datenbytes D8, D6, 37 und 0 gesendet wird.

1.2.7 Analyse von Signalverläufen im Grafikfenster

Während das Datenfenster momentane Signalwerte anzeigt, können Sie im Grafikfenster den zeitlichen Verlauf von Signalwerten darstellen. Nach Messungsende stehen zur Untersuchung der Signalverläufe komfortable Analysefunktionen bereit.

Aufgabe 4: Konfigurieren Sie das Grafikfenster so, daß die Signalwerte der im Sendezweig erzeugten Botschaft 3FC (hex) angezeigt werden.

Auch die zweite im Sendezweig erzeugte Botschaft ist in der zugeordneten Datenbasis beschrieben. In der Datenbasis erkennen Sie, daß dem Identifier 3FC der symbolische Name *GearBoxInfo* mit den Signalen *Gear*, *ShiftRequest* und *EcoMode* zugeordnet ist.

Sie können den zeitlichen Verlauf dieser Signale jetzt im Grafikfenster beobachten. Das Grafikfenster läßt sich genau wie das Datenfenster konfigurieren. Auch hier öffnen Sie den Konfigurationsdialog für Signale im Kontextmenü des Fensters. Wählen Sie im Signalauswahldialog die 3 Signale der Botschaft *GearBoxInfo* aus. Im Grafikfenster erkennen Sie, daß die Signale nun in die Legende auf der linken Fensterseite eingetragen sind. Nach Start der Messung beoachten Sie, daß das Signal *Gear* zyklisch die Werte 1 bis 5 annimmt, während die anderen beiden Signale zeitlich konstant bleiben.

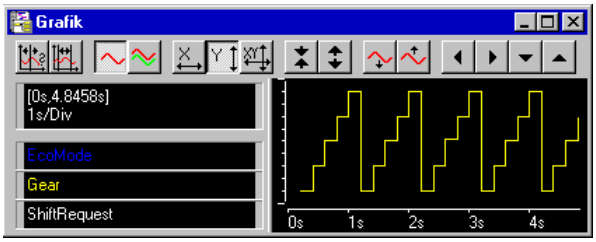


Abbildung 20: Grafikfenster

Dies entspricht den fünf Werten, die Sie in Aufgabe 2 in den Generatorblock eingetragen haben. Nach dem Ende der Messung bleiben die Werte im Grafikfenster erhalten. Die Meßfunktionen, die Ihnen das Fenster zur nachträglichen Analyse von Signalverläufen bietet, sind in Abschnitt 2.5.3.5ff beschrieben.

1.2.8 Einsatz der Datenbasis beim Senden von Botschaften

Bis jetzt haben Sie die symbolische Datenbasis eingesetzt, um Signalwerte zu beobachten. Die Anwendungsmöglichkeiten reichen darüber jedoch weit hinaus. Doppelklicken Sie beispielsweise auf den Generatorblock aus Aufgabe 1, um die Sendeliste zu öffnen. Statt des Identifiers, den Sie vorhin in die Sendeliste eingetragen haben, erkennen Sie in der ersten Spalte nun den zugehörigen symbolischen Namen. Tatsächlich können Sie nun über die Schaltfläche *Symbol...* eine Botschaft direkt aus der Datenbasis in die Sendeliste eintragen, ohne mit dem Identifier arbeiten zu müssen.

Auch die Signalwerte lassen sich nun direkt in der Sendeliste bearbeiten. Wählen Sie die erste Zeile der Sendeliste aus und betätigen Sie dann die Schaltfläche *Signal...*. Im Wertedialog können Sie nun direkt die Signalwerte eingeben. Sie erkennen auch hier noch einmal, daß die Bytewerte D8, D6, 37 und 0 aus der ersten Zeile den Signalwerten EngSpeed = 5500 rpm, EngTemp = 60 Grad Celsius und IdleRunnning = 0 entsprechen.

Abbildung 21: Wertedialog

Wenn Sie nun beispielsweise den Wert von EngSpeed auf 1000 rpm setzten, verwendet der Generatorblock automatisch die Informationen der Datenbasis, um daraus die entsprechenden Datenbytes (10,27,37 und 0) zu berechnen.

1.2.9 Analyse einer Motorraumsimulation

Zum Lieferumfang des CANalyzers gehören einige Beispielkonfigurationen, die Sie bei der Inbetriebnahme unterstützen. Im Verzeichnis `DEMO_CL` finden Sie einen CAPL-Programmblock, der einen Teil der Datenkommunikation auf Motorraumbus eines Kraftfahrzeugs simuliert. Die dem Modell zugrundeliegende Datenbasis `MOTBUS.DBC` haben Sie Ihrer Konfiguration bereits im Aufgabe 3 hinzugefügt.

Aufgabe 5: Das CAPL-Programm `MOTBUS.CAN` simuliert die Drehzahl, Geschwindigkeit, Temperatur eines Fahrzeugmotors. Untersuchen Sie diese Signale im Daten und Grafikfenster, während Sie während der laufenden Messung den Gang mit der '+'- bzw. der '-'-Taste schalten.

Um die Aufgabe zu lösen, löschen Sie zunächst beide Generatorblöcke aus dem Sendezweig des CANalyzers und fügen statt dessen das CAPL-Programm `MOTBUS.CAN` aus Ihrem Demo-Verzeichnis `DEMO_CL` in den Sendezweig ein: Wählen Sie dazu das Kontextmenü des Hot-Spots im Sendezweig *CAPL-Programm einfügen* aus. Im Konfigurationsdialog des eingefügten Programmblocks ordnen Sie danach über die Schaltfläche *Datei...* die Programmdatei `DEMO_CL\MOTBUS.CAN` zu. Schließlich müssen Sie das Programm noch compilieren (Menü *Konfiguration|Compiliere alle Knoten*). Der Sendezweig ist damit vorbereitet.

Konfigurieren Sie nun das Daten- und das Grafikfenster mit den Signalen für Drehzahl (*EngSpeed*) und Temperatur (*EngTemp*) auf der Botschaft *EngineData*, dem Geschwindigkeitssignal (*CarSpeed*) auf der Botschaft *ABSData* und dem Gangsignal (*Gear*) auf der Botschaft *GearBoxInfo*.

Wenn Sie die Messung starten, können Sie im Trace-Fenster den Busverkehr unmittelbar beobachten. Die Botschaften *EngSpeed* und *ABSData* werden zyklisch gesendet, während die Botschaft *GearBoxInfo* nur bei Messungsstart und bei jedem Schaltvorgang, d.h. bei Betätigen der '+'- und der '-'-Taste, spontan einmal gesendet wird.

Die Signalwerte können Sie im Daten- und im Grafikfenster beobachten: Die Temperatur steigt nach Messungsstart langsam bis zu einem Maximalwert an, während die Geschwindigkeit und die Drehzahl zyklisch zwischen zwei Werten schwanken. Das Verhältnis beider Signalwerte wird bestimmt durch den gewählten Gang.

Eine Einführung in die CAPL-Programmierung und Tips zur Erstellung eines kleinen Beispielprogrammes finden Sie in Abschnitt 3.3

1.2.10 Aufzeichnen einer Messung

Zur Datenaufzeichung verfügt der CANalyzer über umfangreiche Logging-Funktionen. Im Standard-Meßaufbau wird der Logging-Zweig ganz unten im Bild dargestellt. Sie erkennen ihn leicht an der Dateitrommel, der die Logging-Datei symbolisiert. Diese wird während der Messung mit CAN-Daten gefüllt.

Aufgabe 6: Zeichnen Sie den gesamten CAN-Datenverkehr, der bei einer kurzen Messung (ca. 20 sec) vom CAPL-Programm `MOTBUS.CAN` erzeugt wird, im ASCII-Format auf.

Um die Daten, die im Meßaufbau des CANalyzers eintreffen, in eine Datei aufzuzeichnen, aktivieren Sie zunächst den Logging-Zweig. Entfernen Sie dazu die Unterbrechung, die den Logging-Block bei einer neuen Konfiguration von der Datenquelle trennt. Mit dem Kontextmenü der Dateitrommel ganz rechts im Logging-Zweig öffnen Sie den Konfigurationsdialog. Hier tragen Sie den Dateinamen für die Meßaufzeichnung sowie deren Format ein. Wählen Sie hier das ASCII-Format.

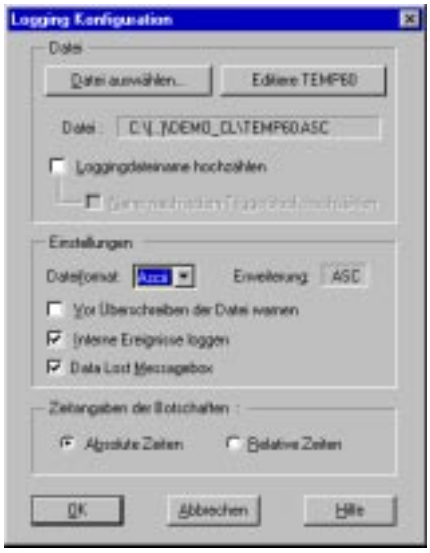


Abbildung 22: Konfigurationsdialog im Logging-Zweig

Aufzeichnungen im Binärformat benötigen weniger Platz auf Ihrer Festplatte, sind aber nicht von normalen Texteditoren lesbar. Der Offline-Modus des CANalyzers bietet Ihnen die gleichen Auswertemöglichkeiten für Aufzeichnungen in beiden Formaten.

Neben der Dateitrommel können Sie im Logging-Block Triggerbedingungen für die Dateiaufzeichnung angeben. Oft ist dies sinnvoll, da nicht der Datenverkehr auf dem CAN-Bus während der gesamten Messung interessiert, sondern nur bestimmte zeitliche Intervalle etwa bei unplausiblen Signalwerten oder beim Auftreten von Error-Frames.

Wie Sie Triggerbedingungen festlegen und Zeitfenster um diese Bedingungen setzten, ist in Abschnitt 2.7 beschrieben. Zum Aufzeichen der gesamten Messung reicht es, im Trigger-Konfigurationsdialog den Modus von *Normal* auf *Gesamte Messung* zu setzen.

Verlassen Sie den Dialog mit *OK* und starten Sie danach die Messung, die Sie nach 20 Sekunden wieder stoppen. Mit einem Doppelklick auf die Logging-Dateitrommel öffnen Sie nun die aufgezeichnete ASCII-Datei. Neben den geloggten Botschaften, die hier genauso dargestellt werden, wie im Trace-Fenster, erkennen Sie, daß auch Statistikinformationen mit aufgezeichnet worden sind. Diese Zeilen entsprechen genau den Informationen, die während einer Messung im Busstatistikfenster dargestellt werden.

1.2.11 Auswerten einer Logging-Datei

Logging-Dateien im ASCII-Format lassen sich zwar mit Texteditoren betrachten. Oft ist es aber sinnvoller, die Möglichkeiten auszunutzen, die der CANalyzer zur Offline-Analyse von Logging-Dateien bereitstellt.

Aufgabe 7: Spielen Sie die der letzten Aufgabe aufgezeichnete Logging-Datei im Offline-Modus des CANalyzers ab und beobachten Sie den Signalverlauf im Grafikfenster.

Um die Aufgabe zu lösen, schalten Sie den CANalyzer zunächst in den Offline-Modus. Im Hauptmenü *Modus* finden Sie dazu die beiden Einträge *zu Offline* und *zu Offline (Kopie)*. Da Sie die im Online-Modus vorbereiteten Einstellungen für das Grafikfenster hier verwenden können, bietet es sich an, alle Konfigurationseinstellungen des Analysezweiges mit *zu Offline Kopie* in den Offline-Modus zu übernehmen.

Im Meßaufbau wird nun statt des PC-Kartensymbols eine Dateitrommel als Datenquelle dargestellt. Der Sendezweig fehlt hier natürlich, während alle Einstellungen des Analysezweiges aus dem Online-Modus übernommen sind. Konfigurieren Sie die Datenquelle, indem Sie auf die Dateitrommel links im Meßaufbau doppelklicken und den Namen der in der letzten Aufgabe gewählten Logging-Datei angeben.

Mit der F9-Taste können Sie nun die Messung abspielen. Im Gegensatz zum Online-Modus bietet Ihnen der CANalyzer hier zusätzlich die Möglichkeit die Messung verlangsamt (Menü *Start|Animate* bzw. F8-Taste) bzw. im Einzelschrittbetrieb (Menü *Start|Step* bzw. F7-Taste) abzuspielen.

Bis auf den Sendezweig stehen Ihnen im Offline-Modus die gleichen Analysefunktionen wie im Online-Modus zur Verfügung. So werden die aufgezeichneten Daten busnah im Trace-Fenster dargestellt, während Sie im Grafikfenster die Signalverläufe der Aufzeichnung beobachten können.

Natürlich können Sie in den Meßaufbau auch Filter oder CAPL-Programme einfügen, um die Daten weiter zu reduzieren bzw. zusätzliche benutzerdefinierte Analysefunktionen einzuführen.

1.2.12 Hinweise für das Bearbeiten eigener Aufgabenstellungen

Diese kleine Tour sollte Sie mit den grundlegenden Bedienkonzepten und wichtigsten Features des CANalyzers vertraut machen. Bewußt wurde an dieser Stelle auf eine Einführung in die Programmiersprache CAPL verzichtet. Falls Sie CAPL zum ersten Mal einsetzten, etwa um eigene Analysefunktionen zu schreiben oder um das Busverhalten eines Steuergerätes zu simulieren - das CAPL-Programm `MOTBUS.CAN` aus der Aufgabe 6 ist hierfür ein Beispiel -, finden Sie in den Abschnitten 3.2 und 3.3 eine kurze Einführung in CAPL.

Denken Sie daran, daß das Meßaufbaufenster des CANalyzers den Datenflußplan für Ihre aktuelle Meßaufgabe darstellt. Abgesehen vom Zuordnen einer Datenbasis können Sie alle Einstellungen direkt in diesem Fenster vornehmen: für die Datenquelle auf der linken Seite, über den Sendezweig bis hin zu den Auswerteblöcken rechts im Fenster. Das Kontextmenü aller Objekte des Meßaufbaus erreichen Sie stets über die rechte Maustaste.

Alle Daten, die in einen Auswerteblock gelangen, werden - abgesehen vom Logging-Block - im zugehörigen Fenster dargestellt. Die Auswertefenster lassen sich ebenfalls über die rechte Maustaste konfigurieren. Sie können alle Konfigurationseinstellungen in einer Konfigurationsdatei abspeichern. Laden Sie einfach eine so vorbereitete Konfigurationsdatei, um den CANalyzer erneut für eine Meßaufgabe vorzubereiten.

Sowohl zu den hier nur kurz angesprochenen Sende und Analysefunktionen des CANalyzers als auch zur CAPL-Programmierung finden Sie im Handbuch ausführliche Erläuterungen. Die kontextsensitive CANalyzer-Hilfe (F1-Taste) beschreibt alle Menüpunkte und erklärt Ihnen die Anzeige- und Bedienelemente aller Dialoge.

1.3 Bedienhinweise

In diesem Abschnitt werden die grundlegenden Bedienverfahren für den CANalyzer erläutert. Falls Sie zum ersten Mal unter Windows arbeiten, sollten Sie sich zunächst mit der Grundlagen der Bedienung von Windows-Anwendungen vertraut machen.

Grundsätzlich läßt sich der CANalyzer sowohl über die Maus als auch über die Tastatur bedienen. So können Sie beispielsweise im Hauptmenü einen Eintrag auswählen, indem Sie mit der linken Maustaste darauf klicken. Wenn Sie in dem darauf erscheinenden Untermenü erneut auf einen Eintrag klicken, so wird die betreffende Aktion ausgeführt.

Alternativ dazu läßt sich das Hauptmenü auch durch Drücken der <Alt>-Taste aktivieren. Sie können nun mit den Cursortasten (<Pfeil rechts>, <Pfeil links>, <Pfeil oben> und <Pfeil unten> einen Eintrag auswählen und mit der Eingabetaste die zugehörige Aktion ausführen.

Sie können einen ausgewählten Menüeintrag wieder deaktivieren, indem Sie <ESC> drücken oder mit der Maustaste außerhalb der Menüfläche klicken.

Alle Fenster der Applikation lassen sich jederzeit, d.h. auch während der Messung, verschieben, vergrößern, verkleinern, öffnen und wieder schließen. Zum Verschieben ziehen (= linke Maustaste drücken und während des Verschiebens gedrückt halten) Sie die Titelleiste des betreffenden Fensters einfach an die neue Position. Um die Größe des Fensters zu verändern, ziehen Sie an den Seiten bzw. Ecken des Fensters.

Alternativ können Sie diese Aktionen auch mit der Tastatur nach Aufruf des Systemmenüs (Drücken von <Alt>-<SPACE> bzw. <Alt>-<->) vornehmen. Weitere Einzelheiten dazu finden Sie in den Windows Handbüchern bzw. der Windows Online-Hilfe.

1.3.1 Menübedienung

Wichtige Grundfunktionen des CANalyzers rufen Sie über das Hauptmenü auf. Die einzelnen Menüpunkte werden in der Online-Hilfe ausführlich beschrieben.

Daneben gibt es in den oben beschriebenen Auswertefenstern und im Datenflußplan im Meßaufbaufenster weitere kontextsensitive Menüs, die es ermöglichen, bestimmte Objekte gezielt zu konfigurieren. Diese Menüs werden durch *Klicken mit der rechten Maustaste* im aktiven Fenster bzw. im Meßaufbaufenster auf den aktiven Block geöffnet. Über die Tastatur läßt sich dies durch Drücken von <F10> erreichen.

Die meisten Blöcke im Meßaufbaufenster lassen sich durch Auswählen des ersten Eintrags im Kontextmenü *Konfiguration* parametrieren. Dazu wird der Konfigurationsdialog des Blocks geöffnet. Sie können diesen Dialog auch ohne Umweg über das Kontextmenü ausführen, indem Sie auf den aktiven Block doppelklicken oder die Eingabetaste drücken.

1.3.2 Dialoge

Neben der Kommandoeingabe, die in der Regel über Menüeinträge erfolgt, gibt es noch Parametereingaben. Parameter werden in der Regel über Dialoge vorgeben. Ein Dialog besteht im allgemeinen aus sechs Feldtypen, die jeweils mehrfach auftreten können:

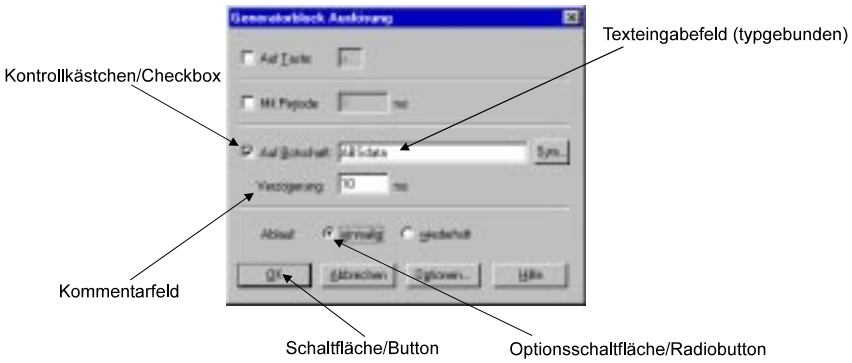


Abbildung 23: Feldtypen

Kommentarfeld	Hier wird erklärt, was eingegeben werden soll. Die Felder verhalten sich beim Anklicken mit der Maus passiv, sie können auch mit der Tastatur nicht angefahren werden.
Texteingabefeld (typgebunden)	Hier unterscheidet man alphanumerische Felder, z.B. zur Eingabe von Dateinamen und numerische Felder, z.B. zur Eingabe von Integer oder Gleitpunktzahlen.
Dropdown-Liste	Nach dem Klicken auf den Pfeil am rechten Rand des Feldes klappt eine Liste auf aus der Sie einen Wert aus einer vorgegebenen Menge auswählen können.
Optionsschaltfläche/ Radiobutton	Diese Felder repräsentieren sich gegenseitig ausschließende Optionen. Sie können immer nur eine Option wählen. Wenn Sie eine weitere Option wählen, wird die bisherige Auswahl aufgehoben. Die ausgewählte Optionsschaltfläche ist durch einen schwarzen Punkt gekennzeichnet.
Kontrollkästchen/ Checkbox	Ein Kontrollkästchen neben einer Option bedeutet, daß Sie diese Option aktivieren oder deaktivieren können. Sie können dabei beliebig viele Kontrollkästchen aktivieren. Aktivierte Kontrollkästchen sind mit einem „x“ gekennzeichnet.
Schaltfläche/Button	Diese Felder dienen zur Ausführung bestimmter Aktionen, z.B. zum Abschluß der Dialogbox oder zum Öffnen einer untergeordneten Dialogbox.

Alle Dialoge besitzen Schaltflächen mit der Bezeichnung *OK*, *Abbrechen* und *Hilfe*. Wenn Sie *OK* drücken, werden die Einstellungen, die Sie im Dialog vorgenommen haben, in die Konfiguration des jeweiligen Blocks übernommen. Drücken Sie *Abbrechen*, so gehen alle Einstellungen verloren, die Sie seit dem letzten Öffnen des Dialogs vorgenommen haben. Mit dem *Hilfe*-Button erhalten Sie einen Hilfetext zu dem Dialog, den Sie gerade bearbeiten. Nach dem Schließen des Hilfefensters können Sie den Dialog weiter bearbeiten. Alle Einstellungen bleiben erhalten.

Die meisten CANalyzer-Dialoge besitzen weiterhin eine Schaltfläche *Optionen*. Nach Betätigen dieser Schaltfläche erscheint ein weiterer Dialog, mit dem Sie die globalen Einstellungen (dezimale/hexadezimale Zahldarstellung, symbolischer/numerischer Modus) des CANalyzers verändern können.

Beachten Sie, daß die Veränderung der globalen Einstellungen aus einem Konfigurationsdialog heraus, Auswirkungen auf die Datendarstellung in allen Fenstern und Dialogen des Systems hat.

Bei mehreren Eingabe- und Aktionsfeldern in einer Dialogbox muß zunächst das gewünschte Feld angewählt werden. Bei Mausbedienung geschieht das durch Anklicken des betreffenden Feldes. Dabei wird bei Eingabefeldern gleichzeitig der Textcursor an der Mauszeigerposition im Feld plaziert, Kontrollkästchen wechseln ihren Zustand und Aktionsfelder werden ausgeführt. Bei der Tastaturbedienung wird das betreffende Feld mit <Tab> bzw. mit <Shift-Tab> ausgewählt. Kontrollkästchen können danach mit der

Leertaste umgeschaltet werden. Die Eingabetaste schließt die Dialogbox ab bzw. führt ein angewähltes Aktionsfeld aus.

1.3.3 Bedienung des Meßaufbaus

Die Konfigurierung von Messungen und Auswertungen erfolgt vorwiegend im Meßaufbaufenster, das den Datenflußplan des CANalyzers für den jeweiligen Betriebsmodus darstellt.

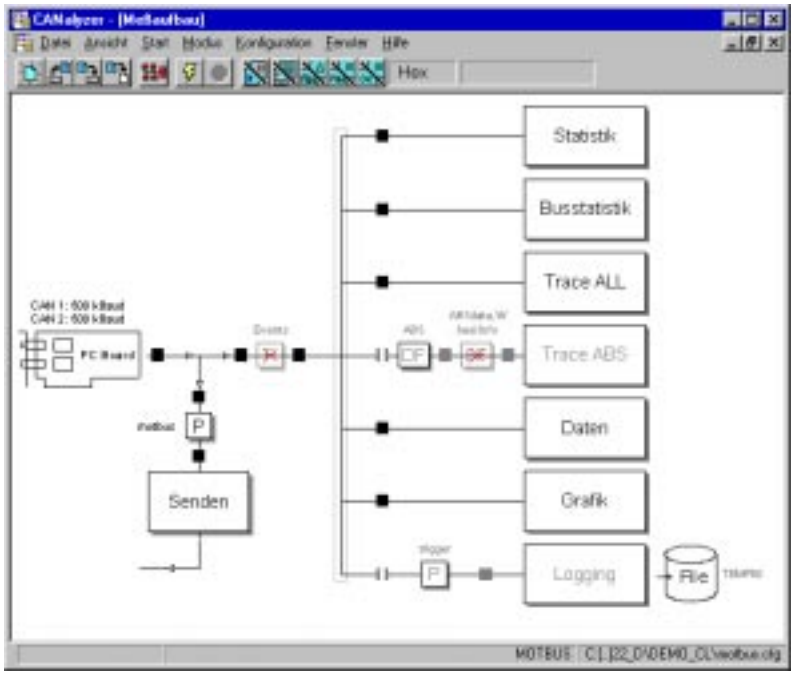


Abbildung 24: CANalyzer Meßaufbau

Mausbedienung

Alle Blöcke und einige Sinnbilder im aktiven Meßaufbaufenster sind maussensitiv. Werden sie durch Klicken mit der linken Maustaste ausgewählt, so wird das so vorgewählte Element durch einen Rahmen als *aktives Element* gekennzeichnet. Durch Klicken mit der rechten Maustaste erscheint dann ein Kontextmenü, über das nach den oben beschriebenen Methoden das Objekt konfiguriert wird. Alternativ dazu kann durch doppeltes Klicken mit der linken Maustaste auch direkt der Konfigurationsdialog des aktiven Blockes aufgerufen werden.

Tastenbedienung

Wird bei aktivem Meßaufbaufenster <Tab>, <Shift-Tab> oder eine der Cursortasten betätigt, so wird der Vorwahlrahmen um das gerade aktive Element weitergeschaltet. <Tab> bewirkt ein Weiterschalten in (<Shift-Tab>: entgegen) der internen Abarbeitungsreihenfolge. Die Cursortasten schalten zum geometrisch nächstliegenden Element in Pfeilrichtung weiter. Durch Betätigen von <F10> erscheint das Kontextmenü des aktiven Elements. Alternativ dazu kann mit der Eingabetaste auch direkt der Konfigurationsdialog des aktiven Blockes aufgerufen werden.

Mit der Leertaste können Sie den vorausgewählten Funktionsblock im Meßaufbau deaktivieren, bzw. nach erneuten Drücken wieder aktivieren.

Mit <Ctrl-F6> und <Ctrl-Shift-F6> können Sie jedes geöffnete Fenster des CANalyzers in den Vordergrund holen und aktivieren.

1.3.4 Das Hilfesystem

Durch Anwahl des Hauptmenüpunktes *Hilfe* erhalten Sie ein Hilfe-Übersichtsfenster, das Grundinformationen und Referenzen zu weiteren Hilfeseiten enthält. Die Referenzen können Sie durch Klicken mit der Maus bzw. durch Weiterschalten mit <TAB> und anschließendem Betätigen der Eingabetaste auswählen.

CAPL-Browser und CANdb-Editor besitzen jeweils ein eigenes Hilfesystem mit einem weiteren Hauptmenüpunkt Hilfe. Aktivieren Sie diese aus dem betreffenden Programm heraus.

Durch Betätigung der <F1>-Taste erhalten Sie für ein zu diesem Zeitpunkt aktiviertes oder vorgewähltes Element die zugehörige Hilfe. Diese kontextsensitive Hilfe ist für alle Dialoge, alle Tochterfenster des Programmes und für alle Menüpunkte sowohl des Haupt- als auch der Kontextmenüs verfügbar.

2 Arbeiten mit dem CANalyzer

2.1 Vorbereiten der Messung

2.1.1 Programmstart

Zum Start des CANalyzers wird das Programm `CANW.EXE` (16-bit Version) bzw. `CANW32.EXE` (32-bit Version) durch Doppelklicken auf das entsprechende Symbol in der CANalyzer-Programmgruppe aufgerufen. Der CANalyzer kann nur dann fehlerfrei arbeiten, wenn das Systemverzeichnis alle notwendigen Dateien enthält und die Hardware richtig installiert wurde (vgl. beiliegende Anleitung zur Hardwareinstallation).

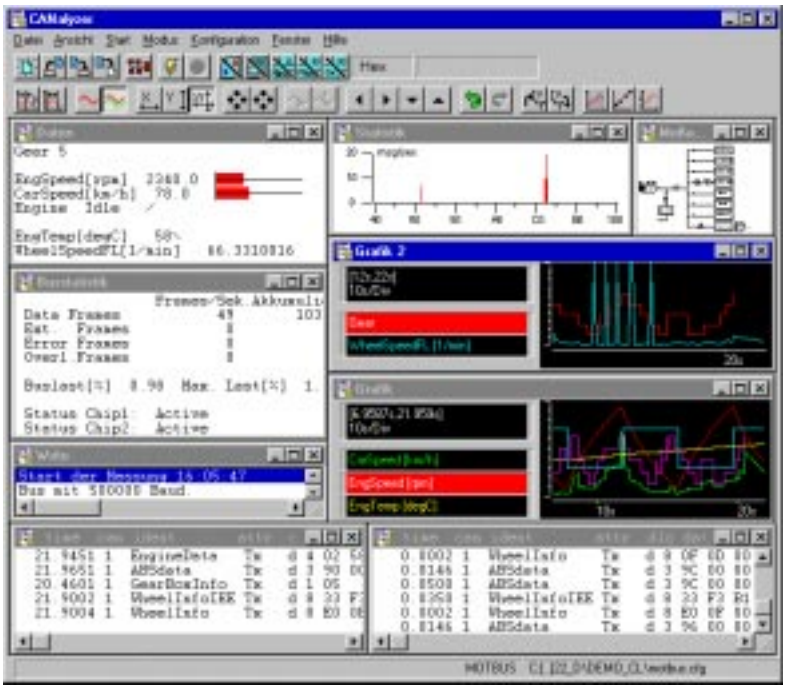


Abbildung 25: CANalyzer während eines Meßvorgangs

Bei Programmstart liest der CANalyzer zunächst Informationen über Hardwareeinstellungen, Anfangspfade, verwendete Editoren, etc. aus der Datei `CAN.INI` im Systemverzeichnis. Danach wird eine Konfigurationsdatei `*.CFG` eingelesen. Diese Datei enthält alle Informationen über die momentane Konfiguration

des CANalyzers und wird bei jedem Programmende nach einer Abfrage automatisch aktualisiert.

Im Programm-Icon können Sie ein Arbeitsverzeichnis angeben. Geben Sie zusätzlich in der Befehlszeile zum Programmnamen noch den Name einer Konfigurationsdatei an, um diese beim Start automatisch zu laden. Damit kann der CANalyzer unter Verwendung mehrerer Icons beim Start unterschiedlich konfiguriert werden. Wurden keine Einträge in der Befehlszeile vorgenommen, wird die jeweils zuletzt geöffnete Konfiguration automatisch geladen.

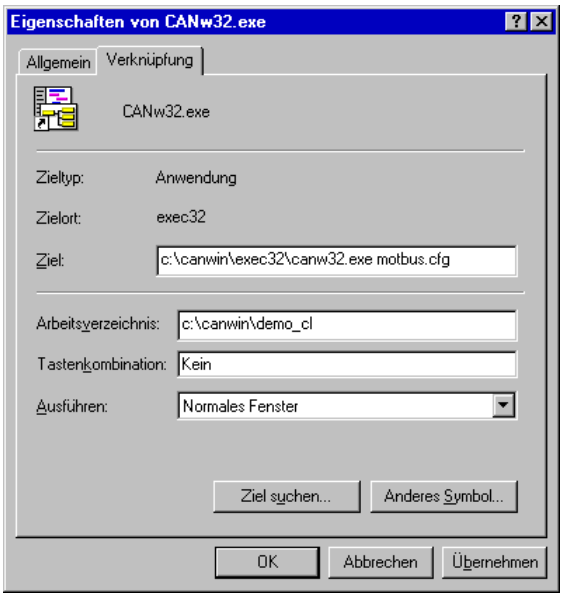


Abbildung 26: Automatisches Laden der Konfiguration `MOTBUS.CFG` bei Programmstart

2.1.2 Der CANalyzer-Bildschirm

Der CANalyzer-Bildschirm besteht aus der Hauptmenüleiste und der Symbolleiste im oberen Teil des Bildschirms, der Statusleiste unten am Bildschirm sowie aus dem Datenflußfenster und den verschiedenen Meßfenstern. Sie gelangen zu allen CANalyzer-Fenstern, indem Sie auf den betreffenden Auswerteblock im Meßaufbau doppelklicken oder indem Sie das Fenster über den Hauptmenüeintrag *Ansicht* auswählen.

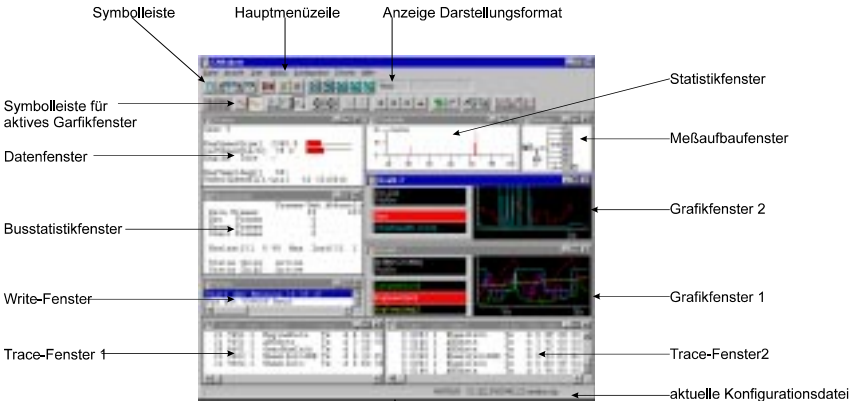


Abbildung 27: Der CANalyzer-Bildschirm

Nach Auswählen eines Eintrags wird das betreffende Fenster aktiviert und im Vordergrund dargestellt.

Die Menüzeile	dient zur Auswahl von Grundfunktionen
Die Symbolleiste	dient zum schnellen Anwählen wichtiger Befehle und enthält außerdem Statusanzeigen für die verwendete Zahlenbasis (dezimal oder hexadezimal) sowie eine Anzeige der Tastatureingaben während der laufenden Messung.
Meßaufbaufenster	Hier wird der Datenfluß im CANalyzer dargestellt. Alle Einstellungen zur Parametrierung einer Messung oder Auswertung werden in diesem Fenster vorgenommen. Die Meßfenster können von hier aus aktiviert werden.
Trace-Fenster	Hier werden die Busaktivitäten protokolliert. Nach Abschluß einer Messung kann in diesem Fenster gescrollt werden.
Statistikfenster	Hier werden die mittleren Sendehäufigkeiten von Botschaften über der Identifier-Achse als Linienspektrum dargestellt. Wahlweise kann auch auf mittlere Sendeabstände umgeschaltet werden. Das Fenster kann für Detailauswertungen gezoomt werden.
Datenfenster	Hier werden vorgewählte Datensegmente von Botschaften dargestellt.
Grafikfenster	Hier wird der zeitliche Verlauf vorgewählter Datensegmente von Botschaften in einem X-Y-Diagramm über der Zeitachse grafisch dargestellt. Nach Messungsende können Sie die Koordinaten von Meßpunkten mit Hilfe von Meßmarkern untersuchen.

Write-Fenster	Hier werden wichtige Informationen über den Verlauf der Messung ausgegeben (z.B. Triggerung der Logging-Funktion). Außerdem werden in dieses Fenster alle Ausgaben geschrieben, die der Anwender in CAPL-Programmen mit der Funktion `write()` absetzt.
Busstatistikfenster	Hier werden hardwarenahe Informationen wie Anzahl der Daten- und Remote-Frames, Error-Frames und Buslast dargestellt. Die Verfügbarkeit dieser Informationen wird durch die verwendete CAN-PC-Karte bestimmt.
Statusleiste	Hier werden die Namen der aktuellen Konfigurationsdatei und der verwendeten Datenbasis angezeigt.

2.1.3 Datenfluß im Meßaufbau

Das Datenflußdiagramm des CANalyzer-Meßaufbaus enthält Datenquellen, Grundfunktionsblöcke, Hot-Spots, eingefügte Funktionsblöcke und Datensenken. Zur Veranschaulichung des Datenflusses sind zwischen den einzelnen Elementen Verbindungsleitungen und Verzweigungen eingezeichnet.

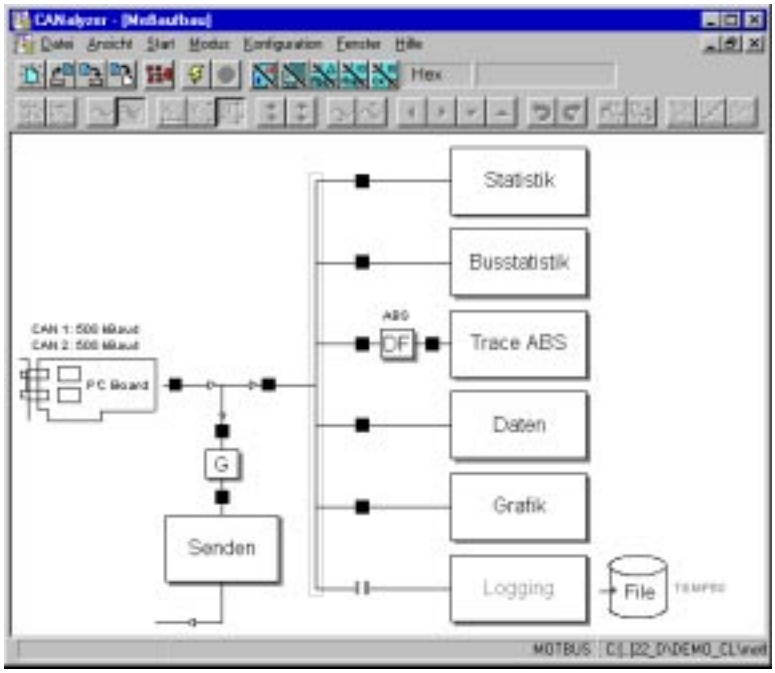


Abbildung 28: CANalyzer Meßaufbau

Als Datenquelle dient im Online-Modus die CAN-PC-Karte, die die CAN-Botschaften am Bus registriert und an den CANalyzer weiterleitet.

Darüber hinaus liefern einige der unterstützten Einsteckkarten weitere Informationen wie z.B. Erkennung von Error- und Overload-Flags, die Werte der Fehlerzähler, die Buslast und externe Triggersignale.

Auf der linken Seite des Meßaufbaus zweigt hinter dem Kartensymbol der Sendezweig des CANalyzers ab. Hier können Sie Funktionsblöcke einfügen, um Daten auf den Bus einzuspeisen. Der Datenfluß im Sendezweig verläuft im Sendezweig stets von oben nach unten; die kleinen Pfeile im Meßaufbau verdeutlichen die Datenflußrichtung.

In den Auswertezweigen werden die Daten von links nach rechts in die Auswerteblöcke im Meßaufbau weitergeleitet, wo sie mit verschiedenen Funktionen visualisiert und analysiert werden können. Vor den Auswerteblöcken können in das Datenflußdiagramm Filter oder benutzerdefinierte Analyseprogramme eingefügt werden. Der Datenfluß läßt sich so auf vielfältige Arten für die jeweilige Aufgabe konfigurieren.

Das Datenflußdiagramm für den Online-Modus enthält auf der rechten Seite Auswerteblöcke

- zur Ausgabe busbezogener Ereignisse mit Zeitinformation auf den Bildschirm (Trace-Block),
- zur Darstellung der Dateninhalte selektierter Botschaften (Datenblock),
- zur grafische Darstellung zeitlicher Signalverläufe (Grafikblock),
- zur Ausgabe der mittleren Botschaftsrate oder des mittleren Sendeabstands zwischen Botschaften mit gleichem Identifier (Statistik),
- zur Anzeige der Raten der Daten-, Remote- Error- und Overload-Frames (Busstatistik) und
- zum Abspeichern der Ereignisse mit Zeitinformation auf Datei für spätere Offline-Auswertungen (Logging)

Zwischen den Funktionsblöcken sind Einfügepunkte (Hot-Spots) angeordnet, an denen Blöcke zur Manipulation des Datenflusses (Filter, Replay, Generatorblock, CAPL-Programmblock mit benutzerdefinierbarer Funktionen) eingefügt werden können. Vor und hinter dem so eingefügten Block erscheinen wieder Hot-Spots, so daß weitere Blöcke eingefügt werden können. An den Hot-Spots kann der Datenfluß auch unterbrochen werden. Eine Beschreibung aller einfügbarer Funktionsblocks finden Sie in Abschnitt 2.6.

Abbildung 28 zeigt eine mögliche Konfiguration des CANalyzers im Online-Modus[1], bei der im Sendezweig ein Generatorblock vorgesehen ist (z.B. zum zyklischen Senden einer oder mehrerer Botschaften). Im Trace-Zweig ist ein Filter eingefügt, so daß nur bestimmte Botschaften im Trace-Fenster angezeigt werden. Der Signalzweig

[1] Der Datenfluß und die Funktionen im Online- und Offline-Modus unterscheiden sich nur an der Datenquelle und im Sendeblock. Eine Beschreibung des Offline-Modus finden Sie in Abschnitt 2.1.3

mit Daten- und Grafikfenster, der Statistikzweig sowie der Busstatistikzweig erhalten jeweils alle Daten, während der Logging-Zweig unterbrochen ist.

Hinweis: Der Datenfluß im Meßaufbau des CANalyzers ist stets gerichtet. Er verläuft von links, ausgehend vom Kartensymbol, nach rechts zu dem Auswertefenstern und im Sendezweig von oben nach unten zum Sendeblock und von dort zur Karte. Botschaften, die im Sendezweig generiert werden (etwa durch einen eingefügten Generatorblock) gelangen also zunächst zum CAN-Controller auf der PC-Karte, von wo aus sie erst bei erfolgreichen Senden wieder in den Meßaufbau links am Kartensymbol eingespeist werden.

2.1.4 Konfiguration des Meßaufbaus

Neben einigen Funktionen wie dem Laden und Speichern von Konfigurationen oder dem Zuordnen von CAN-Datenbasen, die Sie direkt über die Einträge im Hauptmenü aufrufen, dient in erster Linie das Datenflußdiagramm und die Funktionsblöcke im Meßaufbaufenster zur Konfiguration des CANalyzers

Konfigurieren Sie den Meßaufbau, indem Sie einen Block im Datenflußdiagramm aktivieren (Anklicken mit der linken Maustaste oder Verschieben des Auswahlrahmens mit der <TAB>- bzw. den Pfeiltasten). Durch Klicken mit der rechten Maustaste auf den ausgewählten Hot-Spot oder durch Drücken von <F10> erhalten Sie ein Kontextmenü mit allen Konfigurationsmöglichkeiten.

So lassen sich z.B. an den schwarzen quadratischen Einfügepunkten im Datenfluß (Hot-Spots) neue Funktionsblöcke, wie Filter oder Generatorblöcke einfügen oder über das Symbol der Einsteckkarte ganz links im Meßaufbau die CAN-Controller der PC-Karte konfigurieren.

Zum Kopieren und Verschieben von Blöcken stehen Ihnen die Funktionen *Kopieren* und *Ausschneiden* im Kontextmenü jedes Blockes sowie *Einfügen* im Kontextmenü eines Hot-Spots zur Verfügung.

Wollen Sie einen Funktionsblock von der Messung ausschließen, so können Sie ihn vor der Messung mit der Leertaste oder über die Zeile *Knoten aktiv* im Kontexmenü deaktivieren. Dies ist insbesondere dann hilfreich, wenn Sie einen Block bereits konfiguriert haben und ihn nur für bestimmte Messungen ausschalten wollen, ohne ihn zu löschen. Deaktivierte Blöcke werden in gesonderter Form dargestellt, um sie von den aktiven zu unterscheiden. Nach erneutem Betätigen der Leertaste oder nach Anwählen der gleichen Zeile im Kontexmenü wird der Knoten wieder aktiv.

Ein kurzer Blick auf das Meßaufbaufenster gibt Ihnen einerseits einen Überblick über die Konfigurationsmöglichkeiten, die Ihnen der CANalyzer bietet, und zeigt Ihnen andererseits, wie Ihre aktuelle Arbeitskonfiguration aussieht („grafisches Menü“).

Die Darstellung des Meßaufbaus kann in zwei verschiedenen Modi erfolgen:

- Automatisch an die Fenstergröße eingepaßt.
- Feste Vergrößerung mit Bildlaufleisten, falls nötig.

Die Modi wählen Sie über das Kontextmenü des Meßaufbaus aus. Im Vergrößerungsmodus bleiben die Ausmaße des Meßaufbaus erhalten. Ist das Fenster zu klein, um den gesamten Meßaufbau anzuzeigen, so können Sie mit Hilfe der Bildlaufleisten den nicht sichtbaren Bereich in das Fenster scrollen.

2.1.5 Arbeiten mit Auswerteblöcken im Meßaufbau

Alle Auswerteblöcke auf der rechten Seite im Meßaufbau werden untereinander dargestellt. Die Standardauswerteblöcke Statistik und Busstatistik erscheinen dabei stets genau einmal, jeder andere Auswerteblock (Trace, Daten, Grafik und Logging) mindestens einmal.

Um neue Auswerteblöcke in den Meßaufbau einzufügen, klicken Sie mit der rechten Maustaste auf die Verzweigung und wählen das neue Fenster aus dem Kontextmenü aus. Dabei wird der neue Block hinter den letzten des gleichen Typs eingereiht. Er erhält den Standardnamen mit einer fortlaufenden Nummer. Das erste Trace-Fenster heißt also Trace, das zweite erhält den Namen Trace 2 u.s.w.

Um Auswerteblöcke einzufügen, können Sie alternativ dazu auch mit der rechten Maustaste das Kontextmenü eines der Auswerteblöcke öffnen und dort einen neuen Block auswählen.

Sofern mehr als ein Auswerteblock eines der Grundtypen im Meßaufbau vorhanden ist, können Sie diesen über sein Kontextmenü wieder aus dem Meßaufbau entfernen. Dabei wird stets der gesamte Zweig einschließlich aller dort vorhandenen einfügbaren Auswerteblöcke gelöscht.

Um das dem Auswerteblock zugeordnete Fenster zu öffnen, doppelklicken Sie mit der linken Maustaste auf den Block oder wählen Sie im Kontextmenü des Blockes *Fenster anzeigen*. Mehrfach vorhandene Fenster desselben Typs werden im Standardlayout kaskadiert dargestellt.

2.1.6 Messungsstart

Durch Drücken der F9-Taste, durch Anwählen des Hauptmenüeintrags *Start*|*Start* oder durch Betätigen des Startknopfes in der Symbolleiste wird die Messung gestartet. Im Online-Modus können jetzt Daten vom Bus empfangen und ausgewertet bzw. auf den Bus ausgegeben werden. Das Datenflußdiagramm zeigt Ihnen dabei welche Analyse- und Sendeblöcke während der Messung aktiv sind.

Beim Start einer Online-Messung wird zunächst das CAN-Board initialisiert. Falls dies nicht möglich ist, wird eine Fehlermeldung ausgegeben und die Messung abgebrochen.

Während der Messung lassen sich Trace-Block, Datenblock sowie die Skalierung des Statistik- und des Datenfensters frei konfigurieren. Die Einträge in den Kontextmenüs der restlichen Blöcke sind während der Dauer einer Messung jedoch abgeblendet. Sie können diese Blöcke erst wieder nach Beendigung der laufenden Messung parametrieren. Alle Tastatureingaben während der Messung werden direkt an die Funktionsblöcke (CAPL-Programme, Generatorblock, ...) weitergeleitet. Sie werden im entsprechenden Statusfenster in der Symbolleiste angezeigt. Zur Programmbedienung steht lediglich die Taste <Esc> (Messung beenden) sowie alle Tastenkombinationen mit der <Alt>-Taste (Fensterbedienung unter Windows) zur Verfügung.

Sie stoppen die Messung, indem Sie die <ESC>-Taste drücken, den Hauptmenüeintrag *Start*|*Stop* wählen oder indem Sie den Stoppknopf in der Symbolleiste betätigen.

Intern kann die Messung durch Ablauf der Nachlaufzeit nach erfolgter Triggerung oder durch den Aufruf der **`stop()`**-Funktion aus einem CAPL-Programm heraus beendet werden.

Hinweis: Bei hoher Systemlast kann der Stoppvorgang ein gewisse Zeit beanspruchen, da die Nachrichtenpuffer des Systems geleert werden müssen. Durch wiederholtes Stoppen (Doppelklick) werden die gepufferten Daten ignoriert und damit die Messung auch bei hoher Systemlast sofort beendet.

2.1.7 Arbeiten mit Konfigurationen

Alle Einstellungen, die Sie während Ihrer Arbeit am CANalyzer vornehmen (Konfiguration der Meßfenster, des Sendezweigs, der PC-Einsteckkarte, ...) lassen sich in einer Konfigurationsdatei mit der Erweiterung `CFG` abspeichern. Sie können so z.B. mit verschiedenen Konfigurationen arbeiten, um spezifische Messungen und Tests mit dem CANalyzer durchzuführen.

Um Veränderungen einer Konfiguration gezielt in einer neuen Konfigurationsdatei abzuspeichern, wählen Sie im Hauptmenü den Eintrag *Datei|Konfiguration speichern als* an. Mit dem Menüeintrag *Datei|Konfiguration laden* können Sie die Konfigurationsdateien, die Sie zuvor abgespeichert haben, erneut in den CANalyzer laden. Im Demo-Verzeichnis `DEMO_CL` finden Sie einige vorbereitete Demo-Konfigurationen, die Ihnen bei der Inbetriebnahme des CANalyzers und in der Einarbeitungsphase als Vorlage dienen.

Damit Sie den Überblick über die zu Ihrem Projekt gehörigen Dateien (Konfigurationsdateien, Logging-Dateien, CAPL-Programme, Datenbasen, ...) behalten und dieses bei Bedarf auf einem anderen Rechner überspielen können, empfiehlt es sich, für jedes Projekt ein eigenes Projektverzeichnis (in Windows auch Arbeitsverzeichnis genannt) anzulegen. Achten Sie darauf, alle bei Ihrer Arbeit anfallenden Dateien dort abzulegen. Falls Sie mehrere unterschiedliche CAN-Projekte bearbeiten, sind natürlich auch mehrere Projektverzeichnisse sinnvoll. Bei großen Projekten ist es u.U. sogar übersichtlicher, die Datenbasen und Konfigurationsdateien eines Projektes auf verschiedene Unterverzeichnisse zu verteilen.

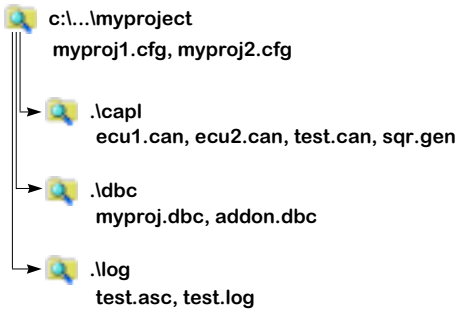


Abbildung 29: Sinnvoller Verzeichnisbaum für ein CANalyzer Projekt

In den Konfigurationsdateien werden auch Referenzen auf andere Projektdateien (wie z.B. auf Datenbasis-Dateien `*.DBC` oder auf CAPL-Programme `*.CAN`) gespeichert. Der CANalyzer arbeitet dabei intern mit *relativen Pfaden*. D.h. referenzierte Dateien werden beim Laden relativ zur Konfigurationsdatei gesucht und eingelesen. Wenn Sie also Ihr Projektverzeichnis mit *allen* zugehörigen Dateien an einen andere Stelle im Verzeichnisbaum bzw. auf eine anderes Laufwerk verschieben, werden die von der Konfigurationsdatei referenzierten Datenbasen und die CAPL-Programme beim nächsten Laden gefunden und korrekt eingelesen.

Die letzten Konfigurationen, mit denen Sie gearbeitet haben, werden im Abschnitt `[LastCANalyzerConfigurations]` der Datei `CAN.INI` gespeichert. Der Menüeintrag *Datei|Alte Konfiguration wählen* bietet Ihnen die Möglichkeit auf diese Konfigurationen zuzugreifen. Wenn Sie in der Kommandozeile keine Startkonfiguration angeben, wird beim Programmstart aus dieser Liste die zuletzt bearbeitete Konfiguration geladen.

Speichern von Konfiguration in früheren Formaten

Ältere CANalyzer Versionen können das aktuelle Konfigurationsdateiformat nicht lesen. Wollen Sie dennoch aktuelle Konfigurationen mit älteren CANalyzer Versionen bearbeiten, so können Sie die Konfigurationen in zu den jeweiligen Versionen kompatiblen Formaten abspeichern.

Wählen Sie die gewünschte Version in der Liste *Dateityp* aus. Beachten Sie jedoch, daß um so mehr Konfigurationsinformationen verloren gehen, je älter das gewählte Format ist. Die aktuell geladene Konfiguration bleibt davon selbstverständlich unberührt.

Die wichtigsten Unterschiede zwischen den Versionen entnehmen Sie der Online-Hilfe

2.1.8 Darstellungsformate

Mit dem Hauptmenüpunkt *Konfiguration|Globale Optionen*erhalten Sie einen Dialog zur Eingabe der Darstellungsformate: Hier können Sie die Zahlenbasis (hex/dec) auswählen, und entscheiden, ob CAN-Botschaften als Identifier oder - sofern Sie eine Datenbasis (vgl. Abschnitt 2.4) zugeordnet haben - symbolisch darstellen lassen möchten.

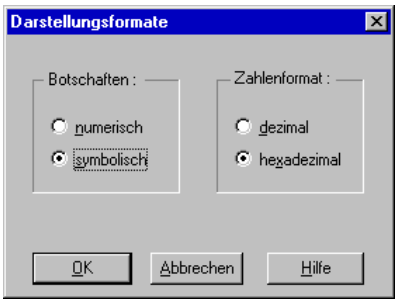


Abbildung 30: Dialog zur Einstellung der Darstellungsformate

Diese Einstellungen wirken sich auf die Darstellungsformate innerhalb des gesamten Programmes aus.

Hinweis: Beachten Sie, daß das Zahlendarstellung im CAPL-Programmen von diesen Einstellungen unbeeinflußt bleibt. Identifier mit vorangestelltem **`0x`** werden hier analog zur Programmiersprache C als Hexadezimalwerte interpretiert, andernfalls nimmt der CAPL-Compiler stets an, daß es sich um Dezimalzahlen handelt.

2.2 Die Schnittstelle zur Hardware

Die Datenquelle des CANalyzers ist im Online-Modus[2] die CAN-PC-Einsteckkarte. Sie liefert die vom CAN-Bus empfangenen Botschaften mit dem Empfangszeitpunkt. Außerdem kann sie auf Wunsch auch Sendeanforderungen mit dem genauen Auftragszeitpunkt quittieren und gesendete Botschaften mit dem genauen Sendezeitpunkt liefern.

Abhängig von der eingesetzten Karte können weitere Ereignisse vom Board geliefert werden. Dazu gehören z.B. die Erkennung von Error- oder Overload-Frames, Messungen der Buslast oder das Eintreffen externer Triggersignale.

Die Karte wird beim Start einer Messung initialisiert. Die dazu notwendigen Parameter müssen Sie dem CANalyzer vor dem Messungsstart mitteilen. Die Parametrierung erfolgt über das Hauptmenü unter *Konfiguration* oder über das Kontextmenü des Kartensinnbilds im Datenflußplan, das Sie durch Anklicken mit der rechten Maustaste oder durch Tastenauswahl mit <F10> erhalten (vgl. Abschnitt 1.3.1). Die Parametrierung ist kartenabhängig. Sie wird im folgenden für die CANcardX beschrieben, die zwei SJA 1000 CAN-Controller enthält. Für andere Boards (CAN-AC2, CANcard, ...) wird die Beschreibung mit der Karte geliefert. Die Bedienung der verschiedenen Dialoge und Bedeutung der Eingabefelder wird in der Online-Hilfe erläutert.

Das Kontextmenü der Karte enthält Menüpunkte zur Parametrierung der Busparameter, zur Einstellung der Akzeptanzfilterung der CAN-Botschaften und zur Eingabe kartenspezifischer Optionen.

[2] Der Offline-Modus zur Analyse von Logging-Dateien wird in Abschnitt 2.7.4 beschrieben.

Bei Hardware-Platformen mit zwei CAN-Controllern sind die Menüpunkte jeweils doppelt vorhanden, um beiden Controller unabhängig voneinander zu parametrieren. Dadurch können auch Anwendungen mit unabhängigen, unterschiedlich schnellen Bussystemen unterstützt werden.

2.2.1 Programmierung der Busparameter

Die Parameter im linken Teil der Dialogbox (*Vorgaben*) geben Sie als Anwender vor, während der rechte Teil (*Ergebnisse*) nur Ausgabefelder enthält (Baudrate, Teilerverhältnis des Vorteilers, die Anzahl der BTL-Zyklen pro Bit und die relative Lage des Abtastpunktes innerhalb des Bits in Prozent) und vom System ausgefüllt wird.

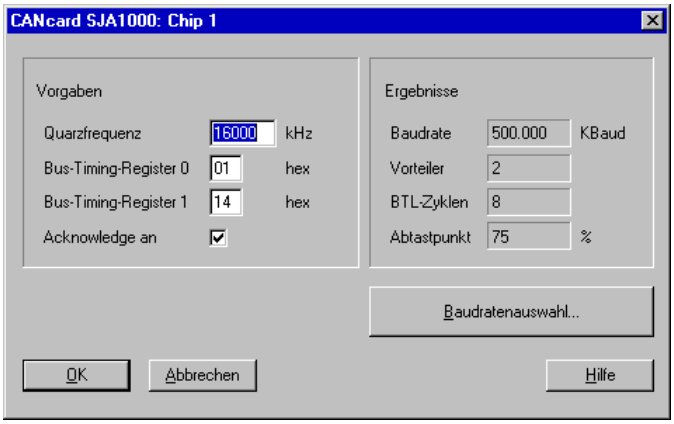


Abbildung 31: Dialogbox Busparameter

Die Schaltfläche *Baudratenauswahl* stellt eine Berechnungshilfe für die BTR-Programmierung dar. Es erscheint ein Dialog, in dem die gewünschte Baudrate und die Anzahl der Samples (1 oder 3) eingetragen werden kann. Auf Betätigung der Schaltfläche *Aktualisieren* wird daraufhin die Listbox mit geeigneten Registerparametersätzen gefüllt. Zur Kontrolle stehen unterhalb der Listbox noch einmal die für diese Listbox gültigen Werte. Wurden links von der Hardware nicht erfüllbare Werte eingetragen (z.B. eine zu hohe Baudrate), so werden Sie bei der Aktion *Aktualisieren* darauf hingewiesen. Es wird Ihnen dabei die nächste gültige Baudrate vorgeschlagen. Sie können den Vorschlag mit *JA* akzeptieren, oder mit *NEIN* ablehnen und damit den Vorgang *Aktualisieren* abbrechen. Um einen der Parametersätze aus der Listbox zu übernehmen, wählen Sie bitte die Zeile an (mit der Tastatur oder einem Mausklick) und wählen Sie den Button *OK*. Mit dem Button *Abbrechen* können Sie den Dialog verlassen, ohne einen Parametersatz zu übernehmen.

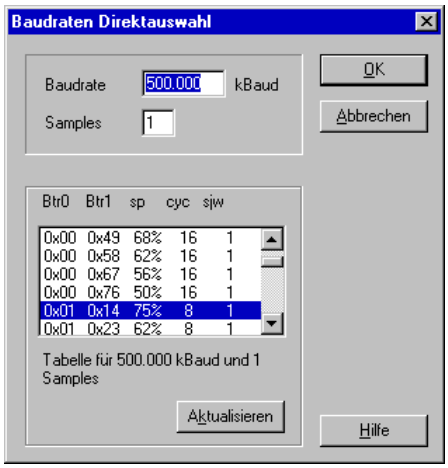


Abbildung 32: Dialog Baudraten Direktauswahl

Die beiden Bus-Timing-Register bestimmen, wie sich ein einzelnes Bit des seriellen Bitstroms auf dem Bus zusammensetzt. Welche Werte hier einzutragen sind, entnehmen Sie bitte dem Datenblatt des CAN-Controllers (SJA 1000/82C200/82527/72005). Geben Sie die Werte als Hexadezimalzahlen ein. Dem rechten Teil der Dialogbox können Sie entnehmen, ob die eingegebenen Werte plausibel sind (d.h., ob z.B. die gewünschte Baudrate erreicht wird).

Da der CAN-Controller auf der Einsteckkarte die Schnittstelle zwischen der Analyse-Software und dem CAN-Bus bildet, wird der Bus durch den Meßvorgang beeinflußt. Insbesondere gibt der CAN-Controller für richtig erkannte Botschaften sein Acknowledge, in dem er im entsprechenden Slot der CAN-Nachricht einen dominanten Pegel auf den Bus legt. Um den Einfluß auf das System zu verringern, läßt sich diese Funktionalität beim SJA 1000 Controller explizit abschalten. Beachten Sie jedoch, daß eine Kommunikation über den Bus bei abgeschaltetem Acknowledge nur zustande kommen kann, wenn mindestens ein weiterer Busknoten ein Acknowledge sendet.

Hinweis: Auf der CAN-AC2-Karte (mit 82C200 bzw. i82527 Controller) läßt sich das Acknowledge nur durch Umsetzen des entsprechenden Jumpers abschalten.

2.2.2 Akzeptanzfilterung

Bei allem Basic-CAN-Controllern der PC-Einsteckkarte (SJA 1000/82C200/82527/72005) wird über eine Maske gesteuert, welche Botschaften gesendet und welche empfangen werden können.

Der SJA 1000 besitzt beispielsweise je eine Akzeptanzfilterung für Standard- und Extended-Identifier und erwartet eine Aufteilung in Akzeptanzmaske und -code. Die Akzeptanzmaske gibt an, welches Bit des ID mit dem Akzeptanzcode verglichen werden sollen. Ist das Bit in der Maske 1, ist das entsprechende Bit nicht relevant für den Vergleich. Ist es 0, wird das Bit des ID mit dem entsprechenden Bit des

Akzeptanzcodes verglichen. Sind beide Bits gleich, wird die Botschaft empfangen, andernfalls ausgefiltert. Sowohl Maske als auch Code werden hexadezimal angegeben.

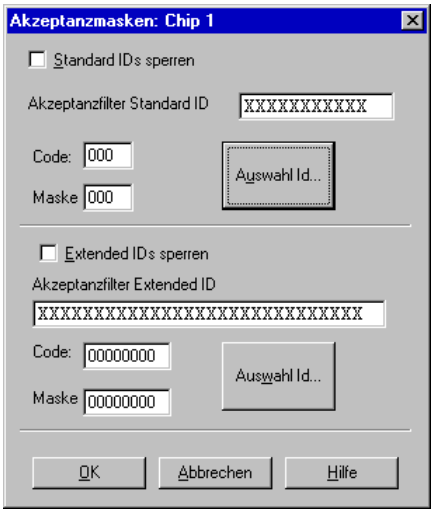


Abbildung 33: Akzeptanzfilterung beim SJA1000 Controller

Statt Maske und Code direkt anzugeben, können Sie die Akzeptanzfilterung im CANalyzer auch über eine logische Maske programmieren (*Akzeptanzfiler Standard ID* bzw. *Akzeptanzfilter Extended ID*). Für jedes Bit in dieser Maske können Sie einen der Werte 0, 1 oder X angeben. Eine auf dem Bus auftretende Botschaft wird nur empfangen, wenn alle mit 0 bzw. 1 angegebenen Bits der Maske mit den entsprechenden Bits der Botschaft übereinstimmen. Die mit X angegebenen Bits werden nicht zum Vergleich herangezogen ("don't care"). Die Werte von Akzeptanzmaske und -code werden nach Eingabe der logischen Maske automatisch berechnet und dargestellt.

Eine ausführlichere Erklärung zur Akzeptanzfilterung aller unterstützten CAN-Controller finden Sie mit Beispielen in der Online-Hilfe.

2.2.3 Karten- und Treiberoptionen

Abbildung 34: Dialog Karten- und Treiberoptionen

Je nach verwendeter CAN-PC-Karte können Sie in diesem Dialog verschiedene Optionen angeben. Hierzu gehören z.B.

- die Meldung des Zeitpunktes der Sendeanforderung (TxRq) zur Messung von Verzögerungszeiten (vgl. Abschnitt 2.3).
- Auswahl, ob die Busstatistik angezeigt werden soll, sowie deren
- Anzeigerate (vgl. Abschnitt 2.5.5).

Die kontextsensitive Hilfe bietet für Ihre Hardwareplatform jeweils genauere Beschreibungen.

2.3 Senden und Empfangen von Daten

2.3.1 Der Sendezweig

Auf der linken Seite des Meßaufbaus zweigt hinter dem Kartensymbol der Sendezweig des CANalyzers ab. Er ist nur im Online-Modus verfügbar und hat dort die Aufgabe, Botschaften, die an seinem Eingang ankommen, an den Kartentreiber als Sendeauftrag weiterzugeben. Der Sendeblock kann selber nicht parametriert werden. Bei der Auswahl des Blocks per Mausklick oder über die Tastatur erscheint weder ein Menü noch eine Dialogbox.

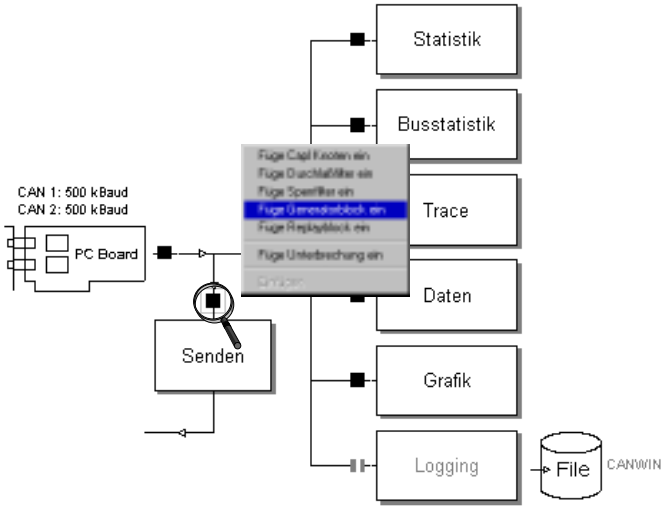


Abbildung 35: Einfügen eines Generatorblockes im Sendezweig

Bei typischen Anwendungen wird im Sendezweig am Hot-Spot vor dem Sendeblock ein Programmblock, ein Generatorblock oder ein Replay-Block eingefügt. Über diesen können Sie spezifizieren, was gesendet werden soll.

Hinweis: Beim Senden aus CAPL-Programmen, Generatorblöcken und Replay-Blöcken können Sie explizit angeben, aus welchen der beiden CAN-Controller (Busanschlüsse) gesendet werden soll. Dies wird vom Kartentreiber entsprechend berücksichtigt.

2.3.2 Die Analysezweige

In den Analysezweigen werden die Daten von links nach rechts in die Auswerteblöcke im Meßaufbau weitergeleitet, wo sie mit verschiedenen Funktionen visualisiert und analysiert werden können. Vor den Auswerteblöcken können in das Datenflußdiagramm Filter oder benutzerdefinierte Analyseprogramme eingefügt werden. Der Datenfluß läßt sich so auf vielfältige Arten für die jeweilige Aufgabe konfigurieren.

Zu jedem Auswerteblock gehört ein Meßfenster, in dem die im Block ankommenden Daten dargestellt werden. Die Funktionen aller Meßfenster werden in den folgenden Abschnitten ausführlich beschrieben. Lediglich dem Logging-Block ist kein eigenes Fenster zugeordnet. Hier können Sie statt dessen eine Logging-Datei zuordnen, um den Busdatenverkehr aufzuzeichnen und anschließend „offline“ zu untersuchen (vgl. Abschnitt 2.7.2).

2.3.3 Botschaftsattribute

Botschaften, die nicht von der CAN-PC-Karte des CANalyzers gesendet wurden (Empfangsbotschaften), erhalten beim Empfang das Attribut Rx und einen

Zeitstempel vom Zeitgeber der Karte. Danach werden sie über den Kartentreiber an den CANalyzer weitergereicht und schließlich in den Auswertefenstern angezeigt. Im Trace-Fenster ist der Zeitstempel und das Botschaftsattribut Rx zu erkennen.

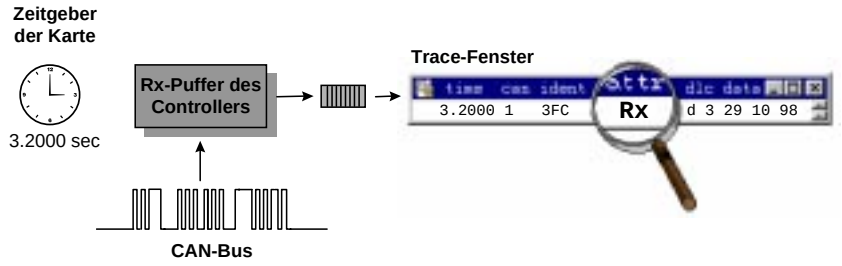


Abbildung 36: Empfangen von Botschaften

Die zu sendenden Botschaften werden im Sendeblock über den Kartentreiber an die CAN-PC-*Karte* durchgereicht. Falls Ihre Hardware die Karten- und Treiberoption *Aktiviere TxRq* im Menüpunkt *Optionen* im Kontextmenü des Einsteckkartensymbols unterstützt und Sie diese aktiviert haben, liefert Ihnen der Treiber den Zeitpunkt des an den CAN-Mikrocontroller vergebenen Sendeauftrags. Im Trace-Fenster sehen Sie so beispielsweise die zu sendende Botschaft mit dem Attribut *TxRq*.

Nach erfolgreicher Sendung wird die Botschaft mit der tatsächlichen Sendezeit und dem Attribut *Tx* zurückgeliefert, so daß die Sendebotschaften mit Sendezeitpunkt beim Tracen dargestellt bzw. mitgeloggt werden können. Diese Botschaften werden, falls sie direkt wieder zum Sendeblock gelangen, nicht erneut gesendet. Dadurch wird vermieden, daß versehentlich Endlosschleifen gebildet werden, die den Bus u.U. stark belasten. Falls bewußt eine Endlosschleife programmiert werden soll, (z.B. als Grundlast) kann dies durch Umkopieren der Botschaft in einem Programmblock erreicht werden.

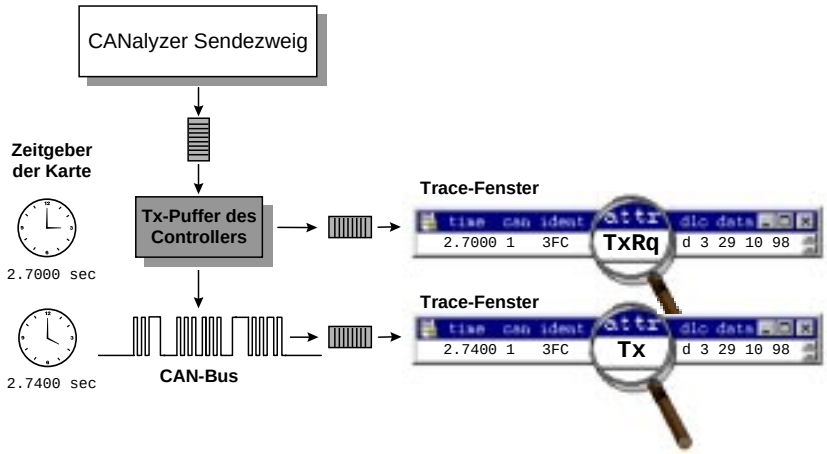


Abbildung 37: Senden von Botschaften

Die *TxRq*-Anzeige ermöglicht Differenzmessungen zwischen Sendeanforderung und Sendezeitpunkt: Die Zeit zwischen der Botschaft mit Tx-Attribut und TxRq-Attribut ist im wesentlichen die Übertragungszeit einer Botschaft, d.h. die Zeit, die der CAN-Controller benötigt, um eine Botschaft vollständig auf den Bus zu legen. Sie ist abhängig von Baudrate und Botschaftslänge. Die Übertragungszeit wächst auch infolge verlorener Arbitrierungsvorgänge, die man zunehmend bei hohen Buslasten für niederprioren Botschaften beobachtet.

Allgemein gilt die Formel

$$t_{Tx} - t_{TxRq} = \text{Übertragungszeit} + \text{Latenzzeit},$$

da zur Übertragungszeit noch die (sehr kleine) Latenzzeit des Kartentreiberinterrupts addiert werden muß.

Hinweis: In Lastsituationen können die Botschaften in den Auswertefenstern unter Umständen leicht verzögert angezeigt werden. Der Zeitstempel der Botschaften bleibt davon jedoch unberührt, da er der Boschaft bereits beim Empfang auf der Karte zugewiesen wird.

2.4 Einsatz von Datenbasen

Bei umfangreichen Untersuchungen am CAN-Bus ist es für den Anwender eine große Hilfe, wenn neben dem busnahen Rohdatenformat mit numerischen Identifiern und Dateninhalten eine symbolische Interpretation des Botschaftsaufkommens zur Verfügung steht.

Der CANalyzer unterstützt die Verwendung symbolischer Datenbasen. Sie stellen diese Information zur Verfügung, indem Sie der aktuellen Konfiguration eine oder mehrere Datenbasen zuordnen (Menüpunkt *Datei|Datenbasis zuordnen*). Danach

können Sie in den Meßfenstern, den einfügbaren Funktionsblöcken und in den CAPL-Programmen auf diese Informationen zugreifen.

2.4.1 Erstellen und Modifizieren von Datenbasen

Zur Eingabe und Änderung der Datenbasen steht Ihnen das Verwaltungsprogramm CANdb zur Verfügung. Es ist im Standardlieferumfang des CANalyzers enthalten. Die 16-bit Version befindet unter `EXEC\CANDB.EXE`; die 32-bit Version finden Sie unter `EXEC32\CANDB32.EXE`.

In einer Datenbasis werden den CAN-Botschaften Namen zugewiesen. Im CANalyzer können Sie die Botschaften dann über diese Namen ansprechen. So erhält man zum Beispiel im Trace-Fenster statt eines Identifiers *100* den Klartext *Motordaten*.

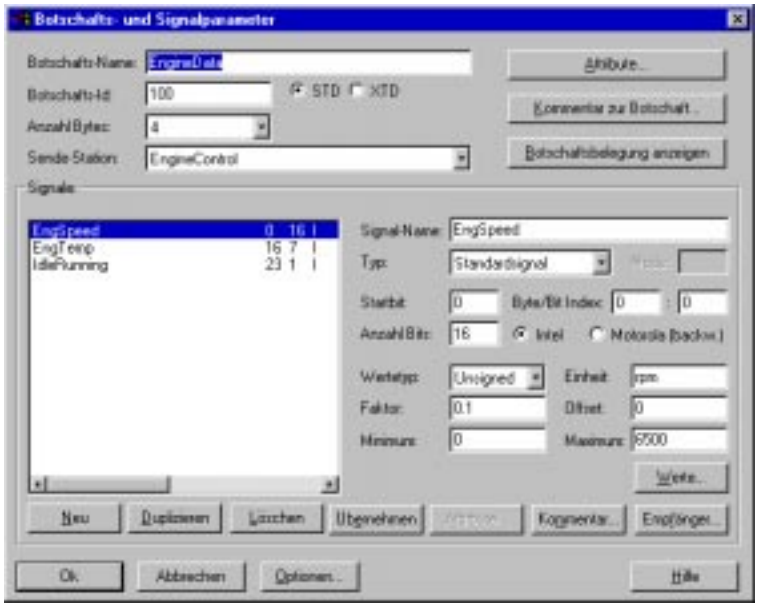


Abbildung 38: Symbolische Beschreibung der Botschaft EngineData in CANdb

Außerdem lassen in den Datenbasen sogenannte *Signale* definieren. Ein Signal ist eine symbolische Beschreibung eines Datensegments innerhalb einer Botschaft. Die Signaldefinition umfaßt neben der Angabe des Datensegments Eigenschaften wie das Maschinenformat (Motorola/Intel), Vorzeichenbehandlung, eine lineare Umrechnungsformel und eine physikalische Maßeinheit. Damit lassen sich im Datenfenster direkt physikalische Größen darstellen wie zum Beispiel

Drehzahl = 810.4 upm.

Weitere Einzelheiten zum Erstellen und Modifizieren einer Datenbasis finden Sie in der Dokumentation zu CANdb bzw. der CANdb-Online-Hilfe. Worauf Sie im CANalyzer beim Arbeiten mit Datenbasen achten sollten, verrät Ihnen Abschnitt 4.3.

2.4.2 Zugriff auf Datenbasisinformationen

Zur Eingabe symbolischer Botschafts- bzw. Signalnamen in Funktionsblöcke befinden sich neben den entsprechenden Texteingabefeldern in der Regel kleine Schaltflächen. Wenn Sie diese Drücken starten Sie den Message-Explorer mit allen in der Datenbasis definierten Symbolen auf, aus dem Sie einen oder (im Filter-Konfigurationsdialog) mehrere Namen auswählen können.

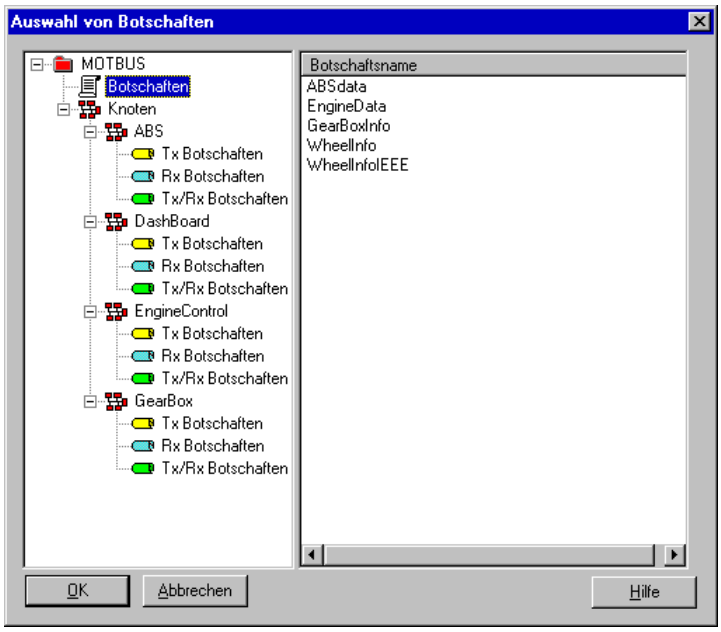


Abbildung 39: Botschafts- und Signal-Explorer

Wenn die Datenbasis vollständig ausgefüllt ist, können Sie in diesem Explorer Signale z.B. für einen Netzknoten, für eine Botschaft oder aber in der Liste aller Signale der Datenbasis suchen.

Weitere Hinweise zur Bedienung des Botschafts- und Signal-Explorers finden Sie in der Online-Hilfe.

2.5 Die Meßfenster

2.5.1 Trace-Fenster

Alle Botschaften, die am Eingang eines Trace-Blocks ankommen, werden ausgewertet und als Informationszeilen im zugehörigen Trace-Fenster dargestellt.

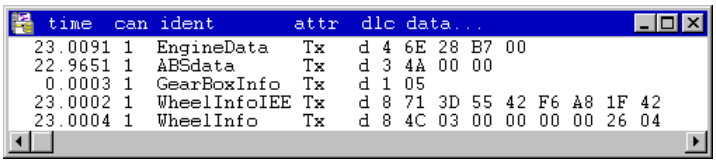

time can ident attr dlc data...
23.0091 1 EngineData Tx d 4 6E 28 B7 00
22.9651 1 ABSdata Tx d 3 4A 00 00
0.0003 1 GearBoxInfo Tx d 1 05
23.0002 1 WheelInfoIEE Tx d 8 71 3D 55 42 F6 A8 1F 42
23.0004 1 WheelInfo Tx d 8 4C 03 00 00 00 00 26 04

Abbildung 40: Trace-Fenster

Abbildung 40 zeigt ein Beispiel für ein Trace-Fenster. Die Abkürzungen im Titelbalken des Fensters sind Überschriften für die darunterliegenden Spalten:

can	Nummer des CAN-Controller-Chips, der die Botschaft geliefert hat. In der Regel wird in dieser Spalte die Zahl 1 oder 2 ausgegeben. Falls die Botschaft von einem CAPL-Programm-Block generiert und dabei die CAN-Nummer nicht explizit festgelegt wurde, wird der Buchstabe C ausgegeben.
ident	Identifier der Botschaft. Entsprechend der Vorwahl erfolgt die Darstellung dezimal oder hexadezimal. Bei einem Extended Identifier wird ein X nachgestellt. Bei Verwendung einer symbolischen Datenbasis erscheinen hier nach Aktivierung der symbolischen Darstellung die Botschaftsnamen.
time	Zeitpunkt zu dem die Information in der CAN-Karte angefallen ist (Empfang, Senden oder Sendeanforderung). Falls die Botschaft von einem CAPL-Programm generiert wurde, wird die im Programm gesetzte Zeit dargestellt. Die Ausgabe erfolgt in Sekunden seit Start der Messung.
attr	Botschaftsattribute Spalte 1: Rx = Empfang, Tx = Senden, TxRq = Sendeanforderung Spalte 2: D(ata) = Datenbotschaft, R(tr) = Remote-Botschaft
dlc	Data Length Code (DLC) der Botschaft
data	Auflistung der Datenbytes der Botschaft. Entsprechend der Vorwahl erfolgt die Darstellung dezimal oder hexadezimal.

Eine Reihe weiterer Ereignisse werden im Trace-Fenster ausgegeben:

Error-Frames:

Bei Auftreten von Error-Frames erscheint eine Meldung im Trace-Fenster.

Trigger-Zeitpunkt:

Im Offline-Modus wird der Trigger-Zeitpunkt der wiedergegebenen Messung mit einer Zeitmarke ausgegeben.

2.5.1.1 Einstellungen für das Trace-Fenster

Sie konfigurieren ein Trace-Fenster über den entsprechenden Menüpunkt im Kontextmenü. Das Trace-Fenster bietet Ihnen verschiedene Ausgabemodi:

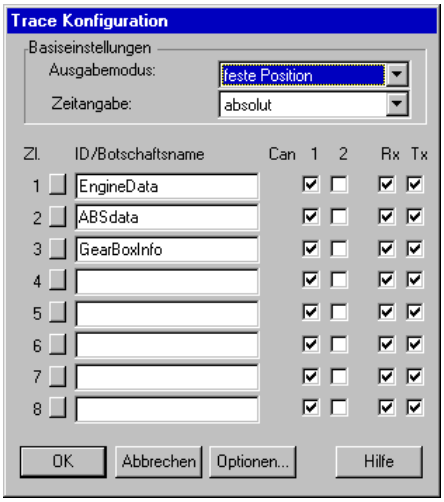


Abbildung 41: Trace-Fenster Konfiguration

Zeitlich folgend

Die ankommenden Botschaften werden zeitlich geordnet dargestellt. Sobald der untere Fensterrand erreicht ist, wird gescrollt, was bei schnellem Busverkehr eine Beobachtung schwierig macht.

Feste Position

Hier werden den Botschaften feste Zeilen zugeordnet. Aktuell ankommende Botschaften überschreiben gleichartige ältere Botschaften auf diesen Zeilen. Dieser Modus hat den Vorteil, daß selbst sehr schneller Busverkehr noch bequem mitverfolgt werden kann und außerdem weniger Rechenzeit benötigt wird.

Die Zuordnung Botschaft zu Zeile kann über den Konfigurationsdialog festgelegt werden. Beispiel: Botschaften mit dem ID 100, die von Controller 1 empfangen werden, sollen stets in Zeile 1 angezeigt werden, Botschaften mit dem ID 200 unabhängig von Controller und Transferrichtung stets in Zeile 2. Nicht explizit zugeordneten Botschaften wird automatisch zur Laufzeit die nächste freie Zeile im Trace-Fenster zugeordnet.

Bei Verwendung einer symbolischen Datenbasis können statt der Identifier auch symbolische Namen verwendet werden. Zur Auswahl drücken Sie die Schaltfläche links von der gewünschte Zeile.

Feste Position, zyklischer Update

Hier werden den Botschaften wie im Modus *Feste Position* feste Zeilen zugeordnet. Das Trace-Fenster wird allerdings nicht mehr bei jeder neu registrierten Botschaft aktualisiert, sondern nur noch periodisch mit einer in der Datei `CAN.INI` konfigurierbaren Zeitkonstanten. Bei hoher Buslast wird somit Rechenzeit eingespart, so daß in diesem Modus Busbeobachtungen mit dem Trace-Fenster auch an leistungsschwächeren Rechnern noch möglich sind (vgl. Abschnitt 4.2).

Sie können jederzeit, also auch während der Messung, zwischen den Darstellungen hin- und herwechseln. So bietet es sich z.B. an, nach der Messung in den chronologischen Ausgabemodus umzuschalten, um in den vom Trace-Fenster empfangenen Botschaften vor- und zurückzublättern. In jedem Trace-Fenster ist dabei ein Puffer eingebaut, in dem die letzten 400 Informationszeilen festgehalten werden.

Hinweis: Das Trace-Fenster läßt sich über das Symbolleisten-Icon während der laufenden Messung anhalten, so daß Sie seinen Inhalt bequem analysieren können, ohne die Messung zu beenden.

Ferner bietet Ihnen das Trace-Fenster zwei verschiedene Zeitformate. Sie können wählen, ob die Zeitstempel der Botschaften absolut, d.h. in Sekunden seit Start der Messung oder aber relativ zur jeweils vorherigen Botschaft angezeigt werden sollen. Im letzteren Fall beziehen sich die Zeitdifferenzen im feststehenden Modus auf jeweils dieselbe Zeile. Sie können in diesem Modus also Sendeabstände zwischen Botschaften mit gleichem Identifier ablesen. Im Modus *zeitlich folgend* werden die Zeitdifferenzen immer zur letzten im Fenster dargestellten Botschaft angezeigt.

2.5.1.2 Einstellungen des Trace-Fensters aus der Symbolleiste

Die Symbolleiste enthält fünf Knöpfe, mit denen Sie die Trace-Fenster direkt konfigurieren können[3]:

Trace-Fenster aktualisieren/anhalten

Trace-Fenster Modus umschalten *zeitlich folgend/feste Position*

Trace-Fenster Modus Zeitdarstellung umschalten *absolut/relativ*

Trace-Fenster Modus Botschaften umschalten *symbolisch/numerisch*

Trace-Fenster Modus Zahlenformat *dezimal/hexadezimal*

Falls Sie mit mehreren Fenstern arbeiten, wirken sich die Aktionen immer auf das aktive Trace-Fenster aus. Falls ein anderes Meßfenster aktiv ist, wird beim Betätigen

[3] Die Knöpfe für Zahlenformat (hex/dez) und Botschaftsformat (numerisch/symbolisch) ändern die betreffenden Einstellungen global für das gesamte Programm.

einer dieser Knöpfe das Fenster des ersten Trace-Blocks aktiviert und die Funktion in diesem Fenster ausgeführt. Wenn Sie nur mit einem Trace-Block arbeiten, wird die Aktion direkt dort ausgeführt, ohne daß das Fenster aktiviert wird.

2.5.1.3 Suchen im Trace-Fenster

Das Trace-Fenster bietet Ihnen im Kontextmenü eine Suchfunktion, mit der Sie nach der Messung im Trace-Fenster nach Botschaften mit bestimmten Eigenschaften suchen können. Die Suchbedingungen, die Sie hier formulieren können, entsprechen weitgehend den Triggerbedingungen für die Dateiaufzeichnung (Logging) und sind in Abschnitt 2.7.5 beschrieben. Die Suchfunktion ist nicht im Ausgabemodus *feste Position* verfügbar. Im Online-Modus erscheinen die zugehörigen Menüeinträge abgeblendet und sind nicht anwählbar.

2.5.1.4 Trace-Watch-Fenster

Das Trace-Watch zeigt bei geöffneter Datenbasis alle Signale, die zur aktuellen Botschaft im Trace-Fenster gehören. Dies erlaubt eine einfache Interpretation der empfangenen Botschaften. Bei einer Änderung der Cursorposition wird der Inhalt des Trace-Watch automatisch aktualisiert.

2.5.1.5 Optimierungen im Trace-Fenster

Da die Trace-Anzeige sehr viel Rechenzeit in Anspruch nimmt, werden die Daten bei hoher Busbelastung u. U. verzögert dargestellt. Die Botschaften tragen in jedem Fall jedoch einen korrekten Zeitstempel, der bereits von der PC-Einsteckkarte vergeben wird. Bei hoher Grundlast kann es vorkommen, daß der Botschaftspuffer überläuft und Botschaften verloren gehen. Im diesem Fall erscheint im Trace-Fenster an der betreffenden Zeile das Überlastzeichen ‘@’. Im Write-Fenster wird zusätzlich eine Warnung ausgegeben. Um einen Pufferüberlauf beim „Mittracen“ zu vermeiden, empfiehlt es sich, bei hohen Grundlasten alle nicht benötigten Zweige (Logging, Statistikfenster, ...) im Meßaufbau abzuschalten. Schalten Sie das Trace-Fenster bei hoher Buslast in den feststehenden Modus mit zyklischen Update um Rechenzeit zu sparen (vgl. hierzu auch Abschnitt 4.2).

Im Offline-Modus spielt die Rechenleistung eine untergeordnete Rolle. Hier kann der Trace-Zweig immer aktiv sein. Es steht auch eine Animate-Funktion zur Verfügung, mit der die gesamte Messung in Zeitlupe wiederholt werden kann, wobei die Einträge ins Trace-Fenster verfolgt werden können.

2.5.2 Datenfenster

Die Datenfenster dienen zur Darstellung von Signalwerten. Als Signale werden Datensegmenten auf einer Botschaft bezeichnet, die bestimmte Informationen tragen (z.B. die Motordrehzahl bei CAN-Bussen in Kraftfahrzeugen). Im Gegensatz zum Trace-Fenster, in dem nur die Botschaften mit den Datenbytes dargestellt werden, können Sie die Werte bestimmter Signale bequem im Datenfenster beobachten. Bei Verwendung einer symbolischen Datenbasis lassen sich die Werte der dort spezifizierten Signale sogar direkt als physikalische Größen darstellen. So läßt sich etwa die Drehzahl in Umdrehungen/min oder die Temperatur in Grad Celsius beobachten.

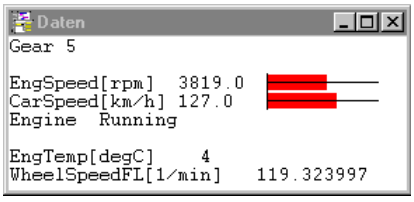


Abbildung 42: Datenfenster

Angezeigt werden stets der Signalname, der im Konfigurationsdialog einstellbar ist, und der zugehörige Wert. Sie können dabei entscheiden, ob der Wert als Rohdatum (hexadezimal oder dezimal), als physikalischer Wert mit zugehöriger Einheit oder als Balkendiagramm dargestellt werden soll.

Die im Fenster dargestellten Signalnamen und –werte sind kontextsensitiv. Wird der Mauszeiger über sie geschoben, so wird das darunterliegende Element mit einem Rahmen gekennzeichnet. Dieses Element läßt sich mit der Maus an jede Stelle des Fensters ziehen, so daß Sie Signale und Werte nach Ihren Erfordernissen gruppieren können. Mit Hilfe des Eintrags *Signale anordnen* im Kontextmenü des Datenfensters lassen sich die dargestellten Elemente wieder in ihre ursprüngliche Anordnung (Signalname links, Wert rechts) zurücksetzen.

2.5.2.1 Konfiguration der Signale

Über das Kontextmenü jedes Datenblocks oder -fensters im Meßaufbau öffnen Sie einen Konfigurationsdialog, in dem Sie die Signale, die im zugehörigen Fenster dargestellt werden sollen, hinzufügen oder löschen sowie ihre Darstellungsparameter angeben.

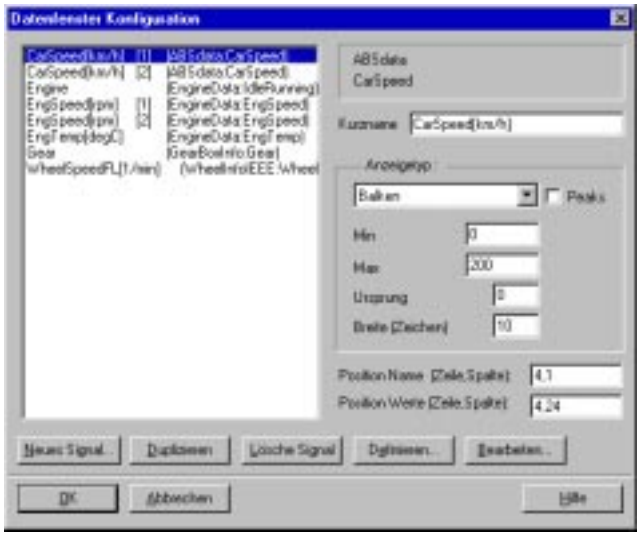


Abbildung 43: Datenfenster Konfiguration

In der Listbox links oben im Dialog werden alle Signale der Signalliste angezeigt. Hier können Sie neue Signale aus der zugeordneten Datenbasis übernehmen. Beim Kopieren wird der Kurzname des Signals so geändert, daß keine Mehrdeutigkeiten entstehen. Der Kurzname erscheint während der Messung als Signalbezeichner neben dem eigentlichen Signalwert.

Mit den Schaltflächen *Definieren* und *Bearbeiten* können Sie eine neue Signalbeschreibung unabhängig von der Datenbasis ins Datenfenster eintragen bzw. eine bestehende verändern.

2.5.2.2 Anzeigetypen

Das Datenfenster verfügt über verschiedene Modi zur Darstellung von Signalwerten. Die Ausgabebreite des Signalwertes wird dabei anhand der Bitlänge und der Umrechnungsformel automatisch berechnet. Der CANalyzer entnimmt die erforderlichen Signaldaten aus der Datenbasis:

Physikalisch

Die Rohdaten werden aus der CAN Botschaft extrahiert, mit der (linearen) Umrechnungsformel skaliert und als dezimale Festkommazahl dargestellt.

Beispiel: Signal T besteht aus 5 Bit, ist ohne Vorzeichen, hat einen Faktor von 10 und einen Offset von 0. Dann ist der mögliche Rohwertebereich 0-31, der physikalische Wertebereich 0-310. Die benötigte Ausgabebreite ist somit 3 Zeichen.

Dezimal

Die Rohdaten werden aus der CAN Botschaft extrahiert und als Dezimalzahl dargestellt.

Beispiel: Signal T besteht aus 5 Bit und ist ohne Vorzeichen. Dann ist der mögliche Rohwertebereich 0-31. Die benötigte Ausgabebreite ist somit 2 Zeichen.

Hexadezimal

Die Rohdaten werden aus der CAN Botschaft extrahiert und als Hexadezimalzahl dargestellt.

Beispiel: Signal T besteht aus 8 Bit. Dann ist der mögliche Rohwertebereich 0-255, in hexadezimaler Darstellung 0-FF. Die benötigte Ausgabebreite ist somit 2 Zeichen.

Bit

Die Rohdaten werden aus der CAN Botschaft extrahiert und als Binärzahl dargestellt.

Beispiel: Signal T besteht aus 8 Bit. Die benötigte Ausgabebreite ist ebenso 8, da alle Bit einzeln dargestellt werden.

Balken

Der physikalische Signalwert wird in Form eines analogen Balkens dargestellt. Für die Darstellung müssen folgende Parameter konfiguriert werden:

Min, Max: Minimaler und maximaler Signalwert, der noch durch den Balken dargestellt werden soll.

Breite: Größe der Balkendarstellung. Die Einheit entspricht der Breite eines Zeichens im Datenfenster.

Mitte: Damit wird die Nullage des Balkens bestimmt.

Mit Hilfe der Parameter Min, Max und Mitte ist es möglich, nur einen Ausschnitt aus dem Wertebereich des Signals darzustellen. Dies kann z.B. der Arbeitspunkt eines zu überwachenden Signals sein.

C-Stil

Diese Option sollten Sie nur dann wählen, wenn Sie mit der Programmiersprache C bzw. C++ vertraut sind. Sie geben in dem Format-Eingabefeld des Dialogs einen Format-String an, der im Datenfenster benutzt wird, um Signalwerte darzustellen.

Die Rohdaten des Signals (ohne Umrechnungsformel) werden aus der CAN Botschaft extrahiert, in einen 32 Bitwert umgewandelt und mit Hilfe der C-Funktion `sprintf()` ausgegeben:

```
sprintf(buffer, FOMATSTRING, (unsigned long)SIGNALVALUE)
```

Der Format-String ist kompatibel zum `printf()`-Format-String von C. Wegen der Transformation auf `unsigned long` muß immer ein L-Modifizierer angegeben werden. Erlaubte Interpretationen sind „lx“, „lX“, „ld“, „f“ und „F“. Sie können außerdem Breiten- und Genauigkeitsmodifizierer verwenden.

Beispiele für Format-Strings: %5.2f, %8lx, %-3ld

Der Datenblock versucht, Breitenspezifizierer im Format-String zu erkennen, um die Ausgabebreite auf dem Bildschirm zu ermitteln. Falls dies nicht möglich ist, wird maximale Breite angenommen. Dies beeinflußt nicht die Ausgabe im Datenfenster.

Beachten Sie bitte, daß Sie als Anwender vollkommen verantwortlich für die Gültigkeit des Format-Strings sind! Ungültige Format-Strings können Folgen bis hin zu Systemabstürzen haben.

2.5.2.3 Aktivitätsanzeige

Die Signalwerte der zuletzt im Datenblock empfangenen Botschaft bleiben im Datenfenster solange sichtbar, bis sie von neuen Werten überschrieben werden. Um bei sich langsam verändernden Signalwerten unterscheiden zu können, ob das Signal bei gleichem Wert aktualisiert wurde oder aber, ob die zugehörige Botschaft nicht mehr gesendet wurde und daher ein nicht mehr aktueller Signalwert im Datenfenster angezeigt wird, verfügt das Datenfenster über eine Aktivitätsanzeige Datenfenster: Wenn eine Botschaft mit unverändertem Signalwert registriert wird, erscheint hinter dem Wert im Datenfenster ein umklappender Schrägstrich (‘/’ bzw.

'\'). Wird dieser Schrägstrich nicht angezeigt, so bedeutet dies, daß der dargestellte konstante Signalwert nicht aktuell ist, da die entsprechende Botschaft nicht empfangen wurde.

2.5.2.4 Peak-Anzeige

Bei sich zeitlich schnell verändernden Signalwerten werden kurzfristige Minima und Maxima leicht übersehen. Um kurzfristige Parks sichtbar zu machen, verfügt das Datenfenster daher über eine Peakdarstellung. Diese erlaubt es Ihnen, in der Balkendarstellung sehr kurzzeitige Signalschwankungen als „Nachleuchten" zu erkennen. Die „Nachleuchtdauer" konfigurieren Sie im Dialog *Timer konfigurieren*, den Sie über das Kontextmenü des Datenfensters aufrufen.

Mit dem Kontrollkästchen *Peaks* können Sie im Konfigurationsdialog des Datenfensters einstellen, ob neben dem aktuellen Wert auch Minimal- und Maximalwerte angezeigt werden sollen: Ist der aktuelle Signalwert größer als das bisherige Maximum, so wird das Maximum sofort auf den neuen Wert aktualisiert. Erst nach einem bestimmten Zeitintervall wird es auf den aktuellen Wert zurückgesetzt. Gleiches gilt sinngemäß für das Minimum.

Die Minimal- und Maximalwerte werden nur bei den Anzeigetypen mit Balken dargestellt. Bei allen anderen Anzeigetypen ändert sich lediglich die Bildschirmfarbe, wenn der Minimal- oder der Maximalwert vom aktuellen Wert abweicht.

2.5.2.5 Optimierung der Datenanzeige

Über das Kontextmenü des Datenfensters öffnen Sie einen Dialog zur Konfiguration des Zeitverhaltens der Anzeige während der Messung.

Die darunterliegenden Optionsschaltflächen um unteren Teil des Dialogs dienen zur Performance-Steigerung des Datenfensters bei großen Datenmengen. Im Normalfall sollten Sie die Default-Option *Sofort Zeichnen* aktiviert lassen. In diesem Fall wird die Datenfensterausgabe nach Empfang einer Botschaft aktualisiert. Zusätzlich wird das Fenster zyklisch alle 2 Sekunden neu gezeichnet.

Um auch große Datenmengen verarbeiten zu können, aktivieren Sie die Option *Nur zyklisch Zeichnen*. Damit wird das Fenster nur noch zyklisch aktualisiert. Dies kann bei hohen Botschaftsraten zu einer wesentlichen Verringerung der Systembelastung führen. Sie sollten die Zykluszeit jedoch nicht größer als 500 ms wählen, um sinnvolle Ausgaben im Datenfenster zu erhalten.

2.5.3 Grafikfenster

Die Grafikfenster dienen zur Darstellung von zeitlichen Signalverläufen. Wie beim Datenblock können Sie sich bei Verwendung einer symbolischen Datenbasis die Werte der dort spezifizierten Signale direkt als physikalische Größen darstellen lassen. So läßt sich etwa die Drehzahlverlauf in Umdrehungen/min oder die zeitliche Entwicklung der Temperatur in Grad Celsius beobachten.

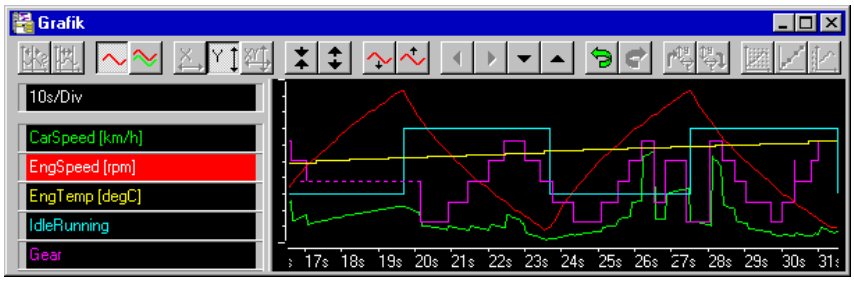


Abbildung 44: Grafikfenster

Die Anzeige erfolgt in einem X-Y-Diagramm über der Zeitachse. Nach Messungsstopp bleiben die Meßdaten im Grafikfenster erhalten und können mit speziellen Meßleisten untersucht werden.

Das Grafikfenster besitzt eine Legende, in der die ausgewählten Signale mit Wertebereich und Farbe angezeigt werden, sowie eine Symbolleiste, mit der Sie die wichtigsten Meßfunktionen bequem aufrufen können. Sowohl die Legende als auch die Symbolleiste sind über das Kontextmenü des Fensters konfigurierbar und lassen sich von dort ein- bzw. ausschalten.

Im Grafikfenster gibt es genau ein aktives Signal, das in der Legende durch inverse Darstellung gekennzeichnet wird. Mit der Tabulatortaste, über Page Up/Down oder durch Anklicken des Signalnamens mit der Maus können Sie das aktive Signal auswählen. Ist der *Einzelsignalmodus* eingeschaltet, so beziehen sich alle Kommandos wie z.B. Messen, Zoomen, Scrollen auf das aktive Signal. Im *Multisignalmodus* beziehen sich die Kommandos auf alle Signale des Grafikfensters.

2.5.3.1 Auswahl der Signale

In Signalauswahldialog des Grafikfensters können Sie –wie im entsprechenden Dialog des Datenfensters– Signale hinzufügen, löschen und ihre Darstellungsparameter angeben. Öffnen Sie den Dialog über das Kontextmenü des Grafikblocks im Meßaufbau, über Kontextmenü des Grafikfensters oder durch einen Doppelklick mit der Maus auf die Legende rechts im Fenster.

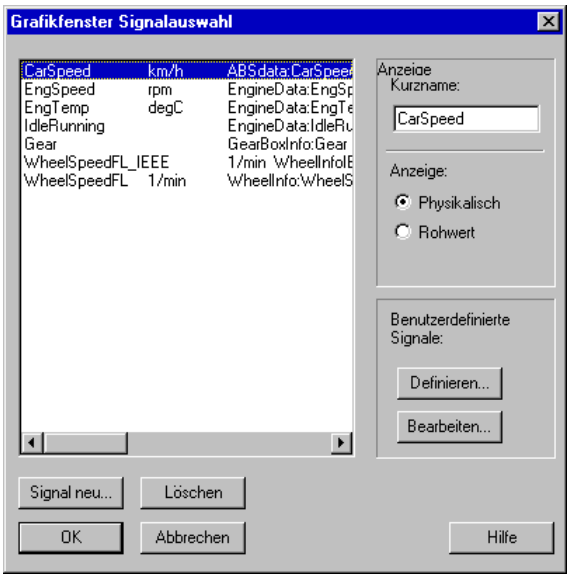


Abbildung 45: Grafikfenster Signalauswahl

In der Listbox links oben im Dialog werden alle Grafikfenstersignale angezeigt. Mit der Schaltfläche *Signal neu...* öffnen Sie zunächst eine Liste zur Auswahl der CANdb Botschaften. In einer zweiten Liste können Sie anschließend die Signale der zunächst von Ihnen ausgewählten Botschaft in die Signalliste des Grafikfensters übernehmen.

Der Konfigurationsdialog erlaubt Ihnen auch unter Umgehung der Datenbasis *benutzerdefinierte Signale* zu erstellen und zu bearbeiten

Geben Sie im Texteingabefeld *Kurzname* einen kurzen, deskriptiven Name für das Signal an, der dann in der Legende des Grafikfensters Signalbezeichner ausgegeben wird.

Als Anzeigemodus stehen Ihnen der physikalische und der dezimale Modus zur Verfügung. Im physikalischen Anzeigemodus werden die Rohdaten aus der CAN Botschaft extrahiert, mit der (linearen) Umrechnungsformel skaliert und als physikalischer Wert im Grafikfenster dargestellt. Die erforderlichen Signaldaten werden aus der Datenbasis ermittelt. Im dezimalen Anzeigetyp werden die Rohdaten dagegen lediglich aus der CAN Botschaft extrahiert und als Dezimalzahl dargestellt. Eine Skalierung mittels der Umrechnungsformel findet nicht statt.

2.5.3.2 Anordnen der Signale

Um konfigurierte Signale von der nächsten Messung auszuschließen können Sie diese vor der Messung deaktivieren. Wählen Sie hierzu das gewünschte Signal aus der Legende aus. In seinem Kontextmenü wählen Sie dann die Funktion

Deaktivieren. Jedes deaktivierte Signal können Sie an derselben Stelle wieder aktivieren.

Sie können darüber hinaus auch die Reihenfolge der Signaleinträge in der Legende des Grafikfensters verändern. Dies ist insbesondere dann nützlich, falls Sie besonders viele Signale konfiguriert haben, das Fenster aber zu klein ist, um alle in der Legende anzuzeigen. Daneben können Sie die deaktivierten Signale an den unteren Rand verschieben, so daß Sie auch bei relativ kleinen Fensterausmaßen möglichst viele aktive Signale noch in der Legende sehen.

Um einen Signaleintrag zu verschieben selektieren Sie ihn mit der Maus und wählen Sie die gewünschte Richtung aus seinem Kontextmenü oder wählen Sie die Tastenkombination <Alt+Pfeil oben> bzw. <Alt+Pfeil unten>.

2.5.3.3 Signal-Layout

Das Signal-Layout, der im Grafikfenster dargestellten Signale läßt sich durch eine Reihe von Funktionen steuern. Grundlegende Einstellungen nehmen Sie im Dialog *Einstellungen des Grafikfensters* vor, den Sie über das Kontextmenü des Fensters oder durch doppeltes Klicken mit der rechten Maustaste auf das Fenster öffnen. Alle Funktionen, mit denen Sie die Signale im Fenster anordnen bzw. nach Messungsende untersuchen können, werden über das Kontextmenü bzw. über die Werkzeugleiste des Grafikfensters geöffnet und sind im nächsten Abschnitt beschrieben.

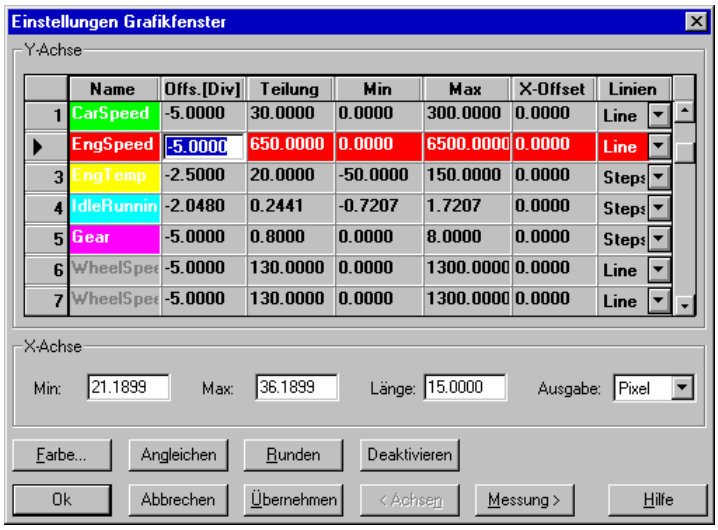


Abbildung 46: Einstellungen der Achsen im Grafikfenster

Für alle Signale, die Sie im Signalkonfigurationsdialog des Grafikfensters angegeben haben, werden automatisch Werte für die Y-Skalierung, die Signalfarbe, die Zeitachse sowie für den Linientyp vergeben. Die Werte für die Y-Skalierung und den

Linientyp werden dabei aus der Datenbasis übernommen. Im Dialog *Einstellungen* des Grafikfensters können Sie alle Einstellungen an Ihre Arbeitserfordernissen anpassen. Der Dialog besteht aus zwei Teilen, den Achseneinstellungen und den Meßeinstellungen. Zwischen beiden Teilen wechseln Sie mit den Aktionsschaltflächen *Messung >* bzw. *< Achsen* hin und her.

In der Signalliste des Dialogs werden alle Signale aufgeführt, die Sie im Signalauswahldialog eingegeben haben.

Linientypen

Der Linientyp in der letzten Spalte bezeichnet die Darstellungsart für die Anzeige einer Signalkurve.

Mit *Linie* werden die Meßpunkte durch eine Strecke verbunden. Diese Darstellung ergibt eine kontinuierliche Kurve. Diese Darstellungsart ist die Defaulteinstellung für die physikalische Darstellung von Signalen, die mindestens ein Byte Länge haben.

Mit der Einstellung *Steps* wird nach einem Meßpunkt eine horizontale Linie bis zur Zeit des nächsten Meßpunktes und von dort eine vertikale Linie zum Meßpunkt ausgegeben. Diese Anzeigeart ist besonders für Digitalsignale geeignet. *Steps* ist die Defaulteinstellung für Signale, die kleiner als 8 Bit sind.

Mit der Einstellung *Horizontal* wird nach einem Meßpunkt eine horizontale Linie bis zur Zeit des nächsten Meßpunktes ausgegeben. Mit *Dots* werden nur die Meßpunkte markiert.

Anzeigemodi

Unter der Signalliste finden Sie vier Eingabefelder zur Konfiguration der Zeitachse. Mit *Ausgabe* definieren Sie dabei den Anzeigemodus.

Fallen mehrere Meßpunkte im Zeitbereich eines Bildschirmpixels zusammen, so werden im Modus *Pixel* jeweils nur Meßpunkte am Rande dieses Bereichs dargestellt. Dies führt zu einer schnelleren Ausgabe bei vielen Meßpunkten auf engem Raum. Unter Umständen werden jedoch einzelne Peaks im Signalverlauf nicht dargestellt, wenn der Meßpunkt mit dem Extremwert im Inneren dieses Bereichs liegt. Im Modus *Voll* werden alle Meßpunkte ausgegeben, auch wenn sie bei der aktuellen Skalierung im Zeitbereich desselben Bildschirmpixels liegen. Bei sehr vielen Meßpunkten auf engem Raum führt dies zu einer geringeren Ausgabegeschwindigkeit, dafür werden jedoch alle Extremwerte des Signalverlaufs dargestellt.

Auf die Meßwerterfassung hat die Einstellung des Ausgabemodus keinerlei Einfluß.

2.5.3.4 Konfiguration der Messung

In den Dialog *Einstellungen im Grafikfenster* gelangen Sie, indem Sie die Aktionsschaltfläche *Messung* aus dem Fensterlayoutdialog betätigen. Hier können Sie das Verhalten des Grafikfensters während der Messung konfigurieren.

Abbildung 47: Meßeinstellungen im Grafikfenster

Wenn Sie unter *Aktionen bei Messungsstart* die Option *Achseneinstellungen nicht verändern* wählen, wird bei Start der Messung der zuletzt eingestellte Ausschnitt des Grafikfensters übernommen. *Mit Zeitachse setzen* wird das Grafikfenster beim Messungsstart auf einen von Ihnen gewählten Zeitausschnitt gesetzt. Sie können mit *Zeitachse und Grundeinstellungen setzen* bei Messungsstart zusätzlich den Y-Achsenausschnitt setzen, den Sie zuvor im Kontextmenü mit *Grundeinstellung speichern* definiert hatten.

Wenn Sie unter *Aktionen bei Messungsstopp* die Option *Zeitachse einpassen* ankreuzen, werden am Ende der Messung die Signalverläufe über die gesamte Meßdauer dargestellt. Sonst bleibt das am Ende der Messung dargestellte Zeitintervall sichtbar.

2.5.3.5 Meß- und Anzeigefunktionen

Sie rufen die Meß- und Anzeigefunktionen des Grafikfensters entweder aus dem Kontextmenü oder durch Betätigen der entsprechenden Schaltfläche in der Symbolleiste auf. Über das Kontextmenü können Sie entscheiden, ob Sie die Symbolleiste im Grafikfenster oder im Hauptfenster anzeigen möchten.

2.5.3.6 Signalmodi

Mit der Funktion *Einzeln* des Kontextmenüs aktivieren Sie den Einzelsignalmodus. Funktionen wie *Vergrößern*, *Verkleinern*, *Grundeinstellung* und *Einpassen* wirken sich im Einzelsignalmodus nur auf das aktive Signal des Grafikfensters aus, das Sie durch Anklicken mit der Maus in der Legende wählen. Im Multisignalmodus (Funktion *Alle*) wirken sich die gleichen Funktionen auf alle Signale des aktiven Grafikfensters aus. Die invertierte Ausgabe des Namens des aktiven Signals in der Legende wird ausgeschaltet.

Hinweis: Es ist immer entweder der Einzelsignalmodus oder der Multisignalmodus aktiv. Die Operationen, welche die Skalierung der Zeitachse ändern, werden - unabhängig von der Einstellung Einzel-/Multisignalmodus - immer für alle Signale durchgeführt, da es im Grafikfenster nur eine Zeitachse für alle Signale gibt.

2.5.3.7 Meßmodi

Punktmeßmodus

Mit der Funktion Meßmarke des Kontextmenüs oder durch den entsprechenden Schalter in der Symbolleiste aktivieren Sie den Punktmeßmodus im Grafikfenster. Falls dieser Modus bereits aktiv ist, wird er durch erneutes Anwählen der Funktion wieder ausgeschaltet.

Im Punktmeßmodus wird eine Meßleiste (vertikale Linie im Fenster) angezeigt, die Sie durch Klicken und Festhalten der linken Maustaste positionieren. Wenn sich der Mauszeiger über der Meßleiste befindet, ändert sich seine Form zu einem horizontalen Doppelpfeil. Wird die Maustaste nicht auf der Meßleiste gedrückt, so wird beim anschließenden Ziehen ein Rechteck aufgezogen. Der Inhalt des Rechtecks wird dann beim Loslassen der Maustaste vergrößert angezeigt (Zoomen). Mit den Tastenkombinationen <Shift-Pfeil links> bzw. <Shift-Pfeil rechts> können Sie die Meßmarke auch per Tastatur verschieben.

Während die Maustaste gedrückt gehalten wird, ist ein kleines Quadrat sichtbar, das den nächstliegenden Meßwert markiert. Der Meßzeitpunkt, der Signalname und der Wert dieses Meßpunktes wird in der Legende oben angezeigt. In der Legende mit den Signalnamen werden die Signalwerte aller Signale zu dem eingestellten Zeitpunkt angezeigt. Die Meßleiste berücksichtigt den Einzel- bzw. Multisignalmodus. Im Einzelsignalmodus springt das Kästchen nur zu Meßpunkten des aktiven Signals, im Multisignalmodus springt das Kästchen zum nächstliegenden Meßpunkt aller Signale.

Differenzmeßmodus

Um Differenzen von Meßwerten zwischen zwei Zeitpunkten auszuwerten, verwenden Sie den Differenzmeßmodus, den Sie über den Eintrag *Differenzmarken* im Kontextmenü oder über die entsprechende Schaltfläche in der Symbolleiste aktivieren.

Im Differenzmeßmodus wird die Meßleiste und eine Differenzleiste (vertikale Linien im Fenster) angezeigt. Wird der Differenzmodus eingeschaltet, so werden die Leisten an ihrer aktuellen Position angezeigt, wenn diese im sichtbaren Bildausschnitt liegt. Ansonsten werden sie in den sichtbaren Bereich verschoben. Durch Klicken und Festhalten der linken Maustaste können Sie die Leisten positionieren. Wenn sich der Mauszeiger über einer Leiste befindet, ändert sich seine Form zu einem horizontalen Doppelpfeil. Wird die Maustaste nicht auf den Leisten gedrückt, so wird beim anschließenden Ziehen ein Rechteck aufgezogen. Der Inhalt des Rechtecks wird dann beim Loslassen der Maustaste vergrößert angezeigt (Zoomen). Die Leisten können jeweils nur im sichtbaren Bereich positioniert werden. Der sichtbare Bereich kann jedoch mit den Pfeiltasten verschoben werden.

Mit den Tastenkombinationen <Strg-Pfeil links> bzw. <Strg-Pfeil rechts> können Sie die Meßmarken auch per Tastatur verschieben.

Während die Taste gedrückt wird, ist ein kleines Quadrat sichtbar, das den nächstliegenden Meßwert markiert. Der Meßzeitpunkt, der Signalname und der absolute Wert (nicht die Differenz) dieses Meßpunktes wird in der Legende oben angezeigt. In der Legende mit den Signalamen werden die Differenzen der Signalwerte aller Signale zwischen den Werten an den eingestellten Zeitpunkten angezeigt. Die beiden Zeitpunkte und die Differenzzeit wird ebenfalls angezeigt. Die Meßleisten berücksichtigen die Einstellung Einzelsignal- bzw. Multisignalmodus. Im Einzelsignalmodus springt das Kästchen nur zu Meßpunkten des aktiven Signals, im Multisignalmodus springt das Kästchen zum nächstliegenden Meßpunkt aller Signale.

2.5.3.8 Anzeigemodi

Für Layoutfunktionen wie *Vergrößern*, *Verkleinern*, *Grundeinstellung* oder *Einpassen* stehen Ihnen im Grafikfenster drei verschiedene Anzeigemodi zur Verfügung:

Im X-Modus wirken sich die Funktionen *Vergrößern*, *Verkleinern*, *Grundeinstellung* und *Einpassen* nur auf die Zeitachse (X-Achse) der Signale aus. Im Y-Modus wirken die gleichen Funktionen nur auf die Werteachse (Y-Achse) aus, während sie im XY-Modus auf beide Achsen gleichzeitig wirken

2.5.3.9 Layoutfunktionen

Das Grafikfenster bietet Ihnen eine Reihe von Funktionen, um das Layout des Fensters zu Verändern. Über das Kontextmenü stehen Ihnen unter anderem folgende Funktionen zur Verfügung:

Alles einpassen

> Unabhängig vom eingestellten Modus wird die Skalierung der Signale wird so eingestellt, daß diese komplett sichtbar sind. Dazu werden die tatsächlichen Minimal- und Maximalwerte jedes Signals sowie der Zeitbereich aller Signale bestimmt und die Skalierung entsprechend eingestellt.

Vergrößern/Verkleinern

> Vergrößerung bzw. Verkleinerung des aktiven Signals (im Einzelsignalmodus) bzw. aller Signale (im Multisignalmodus) um den Faktor 2. Die Größenänderung erfolgt je nach gewähltem Achsenmodus nur für eine Achse (im X-Modus und im Y-Modus) oder aber für beide Achsen gleichzeitig (im X/Y-Modus).
>
> Die Operationen, welche die Skalierung der Zeitachse ändern, werden immer für alle Signale durchgeführt (unabhängig von der Einstellung Einzel-/Multisignalmodus), da es im Grafikfenster nur eine Zeitachse für alle Signale gibt. Die Skalierungen der Achsen können für jedes Signal auch im Dialog *Einstellungen des Grafikfenster* eingestellt werden.

Einpassen

> Die Skalierung der Signale wird so eingestellt, daß diese komplett sichtbar sind. Dazu werden die tatsächlichen Mininal- und Maximalwerte jedes Signals sowie

der Zeitbereich aller Signale bestimmt und die Skalierung entsprechend eingestellt. Dabei werden die aktuellen Modi (X- ,Y- bzw. X/Y-Modus sowie Einzel- und Multisignalmodus) berücksichtigt. Die gesamte Grafik wird so optimal in das Fenster eingepaßt.

Runden

Die Skalierung der Wertebereichsanzeige aller Signale wird gerundet. Dabei wird der aktuelle Teilungswert auf eine gültige Ziffer gerundet (Teilung = $n * 10^x$, n = [1, ..., 9], x : ganze Zahl). Die untere und obere Bereichsgrenze werden auf die Genauigkeit des Teilungswertes gerundet.

Das Runden wirkt immer auf alle Signale des Grafikfensters. Dabei werden die aktuellen Modi (X- ,Y- bzw. XY-Modus) berücksichtigt. Die Skalierung der Signale wird auf ganzzahlige Werte gerundet.

Grundeinstellung/Grundeinstellung speichern

Sie können die Skalierungseinstellung des Grafikfensters (d.h. der angezeigte Zeitbereich aller Signale und die Wertebereiche jedes Signals) als Grundeinstellung speichern. Standardmäßig ist die Grundeinstellung der Zeitbereich von 0 bis 5 Sekunden sowie für jedes Signal der Min-/Max-Wertebereich aus der Datenbasis.

Mit der Funktionen *Grundeinstellung speichern* wird die aktuelle Skalierung des aktiven Grafikfensters als Grundeinstellung gespeichert. Dabei werden die aktuellen Modi (X-, Y- bzw. XY-Modus sowie Einzel-/Multisignalmodus) berücksichtigt. Sie können danach jederzeit mit der Funktion *Grundeinstellungen* alle Skalierungen des Grafikfensters auf diese Grundeinstellungen zurücksetzen.

Markierungen

Dieses Kommando schaltet die Meßpunktmarkierung im Grafikfenster ein bzw. wieder aus. Die Farbe der Meßpunkte entspricht der eingestellten Signalfarbe.

Raster

Dieses Kommando schaltet die Gitterlinien im Grafikfenster ein bzw. wieder aus. Die Gitterlinien entsprechen der Einteilung der Y-Achse.

Y-Achse

Dieses Kommando schaltet die Beschriftung der Y-Achse im aktiven Grafikfenster ein bzw. wieder aus.

Bei ausgeschalteter Beschriftung wird für alle Signale eine Y-Achse mit 10 Unterteilungen angezeigt. In der Legende werden zu jedem Signal die unteren und oberen Werte des Anzeigeausschnitts, sowie der Betrag zwischen zwei Teilstrichen angezeigt.

Wird die Beschriftung eingeschaltet, so wird die Unterteilung der Y-Achse entsprechend dem aktiven Signal automatisch berechnet und in der Farbe des Signals angezeigt.

Die Gitterlinien in der Grafikanzeige entsprechen immer den Teilstrichen der Y-Achse. Ein- bzw. ausgeschaltet werden die Gitterlinien über das Kommando *Raster* im Kontextmenü des Fensters.

Farben

Wählen Sie hier die Hintergrundfarbe des Fensters (Weiß oder Schwarz). Außerdem können Sie von hier den Dialog *Einstellungen* zur Konfiguration der Signalfarben öffnen.

Legende

Wählen Sie mit dieser Funktion, ob die Legende der Signale links im Grafikfenster angezeigt werden soll oder nicht.

2.5.3.10 Export von Signalen

Mit Hilfe dieser Funktion können Sie die Daten eines oder aller Signale des Grafikfensters in eine Datei speichern. Je nach aktuellem Signalmodus (d.h., Einzelsignal- oder Multisignalmodus) wirkt sich der Export entweder auf das aktuell aktive Signal oder auf alle Signale aus. Die Funktion steht nur zur Verfügung, wenn für das aktuelle Signal Daten vorhanden sind.

Für den Export wird das CSV-Format (Comma Separated Values Format) unterstützt. Mögliche Trennzeichen sind Komma, Semikolon, Tabulator [TAB] und Leerzeichen [SPACE].

2.5.3.11 Symbolleiste des Grafikfensters

Das Grafik-Kontexmenü bietet Ihnen die Möglichkeit, die Grafik-Symbolleiste entweder im Hauptfenster oder aber in den Grafikfenstern darzustellen. Wenn Sie die Grafik-Symbolleiste im Hauptfenster anzeigen lassen, wirken sich die Funktionen auf das aktive Grafikfenster aus. Wenn kein Grafikfenster aktiv ist, wird die Grafik-Symbolleiste deaktiviert. Klicken Sie mit der Maus auf ein Grafikfenster, um es zu aktivieren.

2.5.3.12 Optimierung des Grafikfensters

Das Grafikfenster ist ein besonders leistungsstarkes aber auch entsprechend komplexes Fenster im Meßaufbau. Seine Leistungsfähigkeit hängt mit der Grafikkarte Ihres Rechners und dem installierten Grafiktreiber eng zusammen. Unter Umständen können bei der Arbeit mit dem Fenster Effekte auftreten, die die Grafikanzeige beeinträchtigen. Grundsätzlich veranlassen Sie Windows das Fenster neu zu zeichnen, indem Sie z.B. die Größe des Fensters verändern. Unschönheiten bei der Anzeige werden damit oft beseitigt.

Hinweis: Achten Sie während der Messung darauf, daß das Grafikfenster nicht von anderen Meßfenstern, Menüs oder Dialogen überdeckt wird. Die Leistungsfähigkeit der Ausgabe wird von solchen Überdeckungen u.U. stark beeinträchtigt.

Die im Grafikfenster verwendeten leistungsstarken Zeichenalgorithmen benötigen je nach Konfiguration und Buslast eine verhältnismäßig hohe Rechenkapazität. Da die individuellen Anforderungen sehr stark variieren, stellt das Grafikfenster spezielle Konfigurationsmöglichkeiten zur Optimierung der Anzeige bereit.

Öffnen Sie zur Optimierung des Grafikfenster den Konfigurationsdialog *Einstellungen* über das Kontextmenü oder über die Symbolleiste des Fensters. Nach Drücken der Schaltfläche Messung gelangen Sie zu den Meßeinstellungen, wo Sie folgende *Parameter* des Grafikfensters optimieren können:

Puffergröße

Hier bestimmen Sie, über welches Zeitintervall (in Sekunden) der Signalverlauf aller konfigurierten Signale im Grafikfenster gespeichert werden soll. Wenn Sie hier also beispielsweise den Wert 10.0 eingeben, so werden immer die letzten zehn Sekunden ihrer Messung im Grafikfenster abgespeichert und stehen Ihnen nach Messungsende zu weiteren Auswertungen zur Verfügung. Hohe Werte für diesen Parameter führen daher –insbesondere bei der Darstellung vieler Signale– zu einem großen Speicherbedarf, geben Ihnen aber auch die Möglichkeit nach Messungsende den Signalverlauf über einen entsprechend großen Zeitraum zurückzuverfolgen und auszuwerten.

Nach Messungsstopp werden Funktionen, wie z.B. Verschieben oder das Einpassen von Signalen bei großen Signalpuffern u.U. träge, da in diesem Fall große Datenmengen neu gezeichnet werden müssen. Wählen Sie daher den Wert für die Puffergröße stets so klein wie möglich.

Puffergrenze

Zusätzlich zur Puffergröße können Sie eine Puffgrenze in kB angeben. Sie bestimmt die maximale Speicherbelegung durch das Grafikfenster während der Messung. Dies ist vor allem dann sinvoll, wenn Sie eine relativ große Zeitspanne als Puffergröße angegeben haben. Ohne diese maximale Grenze, wird im Verlauf der Messung immer mehr Speicher vom Betriebssystem angefordert. Dabei kann es durch Seitenauslagerungen zu starken Belastungen des Gesamtsystems kommen. Weitere Einzelheiten finden Sie in der Online-Hilfe.

Benutzerdefinierter Refresh

Hier legen Sie fest, wie oft die Anzeige des Grafikfensters aktualisiert werden soll. Kleine Werte haben eine kontinuierliche Darstellung des Signalverlaufs zur Folge, erfordern andererseits aber eine hohe Rechnerleistung und führen bei langsameren Rechnern eventuell zu Lastproblemen. Hohe Refresh-Werte senken den Rechenaufwand insbesondere bei der Darstellung vieler Signale beträchtlich, führen aber zu einer unruhigeren Signalverlaufsanzeige. Geben Sie einen Refresh-Wert nur vor, wenn Ihre Meßkonfiguration besondere Anforderungen (hohe Buslast, gleichzeitige Darstellung vieler Signale, ...) an das Grafikfenster stellt. Bei nicht angekreuzter Refresh-Checkbox wird automatisch ein günstiger Default-Wert vom Grafikfenster bestimmt. Solange während der Messung keine Lastprobleme (vgl. Abschnitt 4.2) auftreten, sollten Sie diese Einstellung nicht verändern.

Scrolling

Wenn die Signalkurven nach dem Start der Messung am rechten Rand des Grafikfensters anstoßen, folgt dieses den Kurven automatisch. Die Zeitachse wird dabei nach links verschoben, um auf der rechten Seite Platz für das

Meßsignal zu schaffen. Sie können dieses als Scrolling bezeichnete Verhalten im Grafikfenster konfigurieren. Als Defaulteinstellung ist dabei das fließende Scrolling aktiviert. Dabei wird die Zeitachse stets nur minimal nach links verschoben, so daß der Eindruck kontinuierlich durchlaufender Signalkurven entsteht.

Um die Zeitachse in Sprüngen zu verschieben und so das Grafikdiagramm zwischen diesen Sprüngen zu fixieren, schalten Sie im Grafikfensterdialog *Einstellungen* das fließende Scrolling ab. In der darunterliegenden Zeile können Sie dann einstellen, um wieviel Prozent des dargestellten Zeitintervalls die Zeitachse verschoben werden soll. Je kleiner der Wert ist, desto gleichmäßiger wird das Bild abgerollt. Entsprechend häufiger wird das Bild aber auch verschoben. Falls das im Grafikfenster dargestellte Zeitintervall kleiner ist, als der von Ihnen im angegebene Wert, wird das gesamte Grafikfenster neu aufgebaut, d.h. das Bild wird um 100% verschoben. Damit wird verhindert, daß das Grafikfenster bei sehr kleinen Zeitintervallen durch zu häufiges Scrolling zu viel Rechenleistung verbraucht.

Das Scrolling-Verfahren wird mit kleineren Zeitintervallen immer rechenaufwändiger, da immer schneller gescrollt werden muß. Es gibt daher einen minimalen Refresh-Zyklus, mit der der Fensterinhalt aktualisiert wird. Liegt das dargestellte Zeitintervall in der Größenordnung dieses Zyklus', so werden die Signalkurven wieder zunehmend ruckartig dargestellt, da der Fensterinhalt nicht immer aktualisiert wird. Sie können den minimalen Refresh-Zyklus in der Datei `CAN.INI` einstellen. Setzen Sie dazu im Abschnitt `[System]` den Wert

```
GraphicWindowMinAutoRefreshrate = <Zykluszeit in ms>
```

Beachten Sie jedoch, daß die Systembelastung wächst, je kleiner Sie diesen Wert wählen. Vor dem Ändern des automatischen Refresh-Zyklus' sollten Sie daher mit Hilfe der Funktion *Benutzerdefinierter Refresh* unbedingt zunächst eine optimale Refresh-Rate ermitteln.

Hinweis: Einige treiberspezifische Probleme bei der Darstellung der Meßleisten und von punktierten Linien, wie sie beim Zeichnen von extrapolierten Signalen und der Gitterlinien des Fensters verwendet werden, können durch spezielle Einstellungen in der Datei `CAN.INI` gelöst werden. Vergleichen Sie dazu die Dokumentation im Abschnitt `[Debug]` der INI-Datei.

2.5.4 Statistik

Der Statistikblock erfüllt zwei unterschiedliche Funktionen. Zum einen werden während einer Messung die mittleren Botschaftsraten bzw. -zeitabstände dargestellt. Dazu wird über einer Identifier-Achse ein Linienhistogramm aufgebaut und ständig aktualisiert. Es wird eine gleitende Mittelwertbildung mit einstellbarer Mittelungszeit verwendet. Zum anderen wird im Hintergrund eine Statistik über alle Busaktionen geführt, deren Ergebnis nach Abschluß einer Messung protokolliert werden kann.

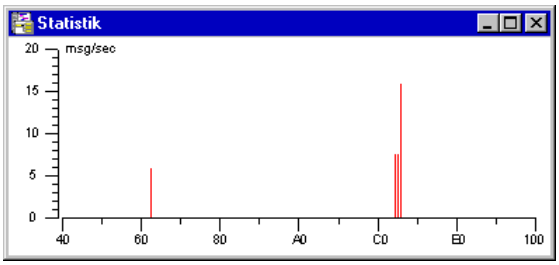


Abbildung 48: Statistikfenster

Das Statistikfenster stellt (siehe Abbildung 48) die mittleren Botschaftsraten dar, die bei Abschluß der Messung vorlagen. Das Write-Fenster enthält das Statistikprotokoll (siehe Abbildung 50).

2.5.4.1 Direktanzeige im Statistikfenster

Während der Messung wird im Statistikfenster entweder der mittlere Sendeabstand oder die mittlere Botschaftsrate dargestellt. Hierzu wird eine gleitende Mittelwertbildung mit einstellbarer Mittelungszeit verwendet. Auf der horizontalen Achse werden die Botschafts-Identifier aufgetragen, auf der vertikalen die entsprechende Rate. Die ID werden aufgespalten nach Herkunft von den Controllern CAN1 und CAN2 sowie nach den Botschaftsattributen Rx und Tx:

	Rx	Tx
CAN1	Grün	Rot
CAN2	Grau	Blau

Im Statistik-Konfigurationsdialog können Sie festlegen, ob das Fensterdiagramm den mittleren Sendeabstand (sec/msg) oder den Kehrwert, die mittlerer Botschaftsrate (msg/sec) darstellen soll. Einstellbar ist ebenfalls die Mittelungszeit Damit geben Sie an, in welchen Zeitabständen die Darstellung aufgefrischt wird. Bei einem niedrigen Wert ist die Mittelung am genauesten, verbraucht aber viel Rechenzeit. Bei höheren Werten hinkt die Statistik in ihrer Darstellung hinterher. Eine Mittelungszeit von ca. 1000 ms erzielt bei Standardanwendungen befriedigende Ergebnisse.

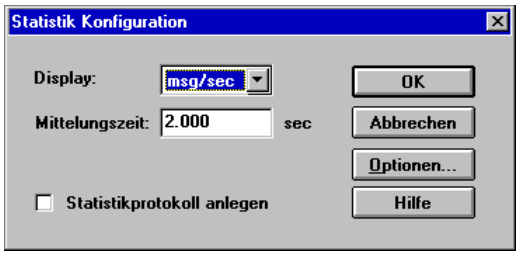


Abbildung 49: Konfiguration des Statistikblocks

Sie können das Statistikfenster über das Kontextmenü skalieren. Die dazu verfügbaren Funktionen wie *Zoom*, *Fit*, *Grundbild* und *Manuelle Skalierung* sind in der Hilfe des CANalyzers ausführlich beschrieben.

Hinweis: Unterstützt die verwendete CAN-Karte Extended Identifier, ist die Funktion *Grundbild* aufgespalten. Es kann dann wahlweise auf den Bereich der Standard Identifier oder auf den gesamten Bereich skaliert werden.

2.5.4.2 Statistikprotokoll

Im Hintergrund wird eine Statistik über alle Busaktionen geführt, deren Ergebnis nach Abschluß einer Messung ins Write-Fenster protokolliert werden kann. Es wird eine nach Botschafts-Identifiern geordnete Liste aufgebaut, in der jeweils getrennt für Empfangsbotschaften, Sendebotschaften, Sendeanforderungen und Sendeverzögerungen folgende Informationen enthalten sind: Anzahl der Botschaften, mittlerer Zeitabstand, Standardabweichung, Minimalabstand, Maximalabstand.

Mit der Option *Statistikprotokoll anlegen* im im Statistik-Konfigurationsdialog werden alle Daten für das Protokoll akkumuliert. Dies verbraucht allerdings Rechenleistung und kann sich bei sehr schnellem Busverkehr bremsend auswirken.

Wenn Sie ein Statistikprotokoll angelegt haben, können Sie dieses nach Ende der Messung im Write-Fenster ausgeben. Wählen Sie dazu im Kontextmenü des Fensters bzw. des Blockes die Funktion *Statistikprotokoll anzeigen*.

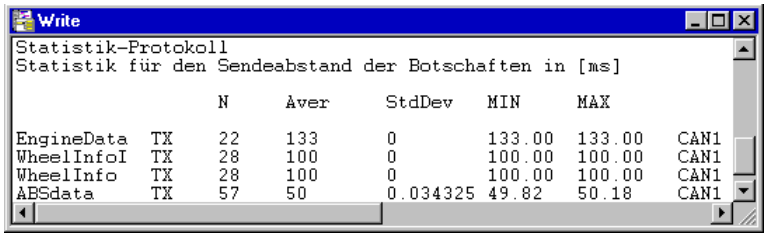

Write
Statistik-Protokoll
Statistik für den Sendeabstand der Botschaften in [ms]
N Aver StdDev MIN MAX
EngineData TX 22 133 0 133.00 133.00 CAN1
WheelInfoI TX 28 100 0 100.00 100.00 CAN1
WheelInfo TX 28 100 0 100.00 100.00 CAN1
ABSdata TX 57 50 0.034325 49.82 50.18 CAN1

Abbildung 50: Statistische Auswertung einer Messung im Statistikprotokoll

Danach wird im Write-Fenster für jeden aufgetretenen ID, aufgeschlüsselt nach den Attributen, die Gesamtzahl der Botschaften, der mittlere Sendeabstand, dessen Standardabweichung, Minimum und Maximum angegeben.

Hinweis: Die Funktion *Statistikprotokoll anzeigen* kann nur dann ausgewählt werden, wenn während der Messung die Option *Statistikprotokoll anlegen* gewählt wurde.

2.5.5 Busstatistik

Der CANalyzer erfaßt über den Kanal 1 Statistikdaten, deren Umfang von der verwendeten Hardware abhängt. Die Ergebnisse werden im Busstatistikfenster dargestellt. Die statistischen Funktionen umfassen dabei die Raten und Anzahl der Daten- und Remote-Frames (11 und 29 Bit), Error-Frames und Overload-Frames. Außerdem werden die aktuelle und die maximale Buslast angezeigt. Der Zustand der CAN-Controller wird als ACTIVE, ERROR PASSIVE oder BUS OFF dargestellt. Für bestimmte Hardware-Platformen wird außerdem der Wert des Tx-Fehlerzählers und des Rx-Fehlerzählers des CAN-Controllers dargestellt.

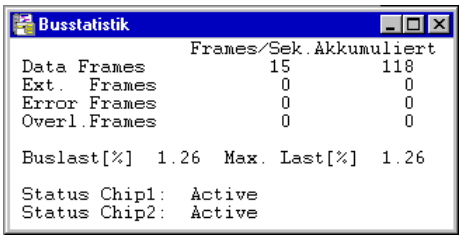


Abbildung 51: Busstatistikfenster

Im Kontextmenü *Optionen* des Kartensymbols im Meßaufbaufenster kann das Zeitintervall eingestellt werden, mit der die Karte Busstatistikinformationen an den CANalyzer weiterreicht. Dieses Intervall bestimmt die Häufigkeit der Buslastmessung und somit auch die Mittelungszeit. Der Standardwert ist 1 Sekunde. Für Messungen mit extremem Datendurchsatz kann die Busstatistik abgeschaltet werden um die Performance zu steigern. Dies wirkt sich vor allem auf den FIFO-Puffer zwischen CANalyzer und CAN-Karte aus. Die Fehlermeldung "Rx-Buffer-Overrun" kann evtl. so vermieden werden.

Busstatistikinformationen werden auch im Logging (vgl. Abschnitt 2.7) registriert. Aktivieren Sie im Konfigurationsdialog der Logging-Datei die Checkbox *Interne Ereignisse loggen*, um die Informationen in der Logging-Datei mitabzuspeichern. Beim Abspielen der Datei im Offline-Modus werden diese Informationen dann wieder im Busstatistikfenster angezeigt. Das Busstatistikfenster bleibt im Offline-Modus leer, wenn die Datenquelle keine Busstatistikinformationen enthält.

2.5.6 Write-Fenster

Das Write-Fenster hat im CANalyzer zwei Funktionen: Zum einen werden hier wichtige Systemmeldungen über den Verlauf der Messung ausgegeben (z.B. Start- und Stoppzeit der Messung, eingestellte Baudrate, Triggerung der Logging-Funktion, Statistikprotokoll nach Abschluß der Messung). Zum anderen werden hier alle

Meldungen ausgegeben, die Sie als Anwender in CAPL-Programmen mit der Funktion **`write()`** absetzen.

Mit dem Kontextmenüeintrag *Kopiere Inhalt ins Clipboard* übernehmen Sie den Inhalt des Write-Fensters in die Zwischenablage. Write-Fenster-Meldungen dienen sowohl als ergänzendes Protokoll zu Ihren Messungen als auch – beim Auftreten von Problemen – als Grundlage für eine Fehleranalyse durch unseren Kundendienst.

Eine Beschreibung der wichtigsten CANalyzer-Systemmeldungen, die ins Write-Fenster ausgegeben werden, finden Sie im Anhang in Abschnitt 5.4

2.5.7 Layout und Fonts der Meßfenster

Die Anordnung der Fenster (Fensterlayout) können Sie über den Hauptmenüpunkt *Fenster* beeinflussen:

Mit *Standard Layout* wird die Standard-Fensteranordung geladen. Der Aufruf dieser Funktion ist sinnvoll, wenn Sie die Fenstergrößen und -positionen verstellt haben und schnell wieder zur Standardeinstellung zurückwechseln möchten. Im Gegensatz zur Funktion *Datei|Konfiguration neu* wirkt sich diese Funktion nur auf die Fensteranordnung aus, alle übrigen Einstellungen bleiben erhalten.

Wählen Sie die Funktion *Benutzerdefiniertes Layout setzen*, um eine Ihren Arbeitserfordernissen entsprechende Fensteranordnung (Layout) zu definieren. Diese Fensteranordnung wird in der Datei `CAN.INI` gespeichert. Mit dem Menüeintrag *Benutzerdefiniertes Layout* können Sie anschließend die Fenster jeder geöffneten Konfiguration entsprechend den mit dieser Funktion gesetzten Einstellungen anordnen.

Im unteren Bereich des Menüs werden alle gerade geöffneten Fenster des CANalyzers angezeigt. Das aktive Fenster ist zusätzlich mit einem „√“ gekennzeichnet.

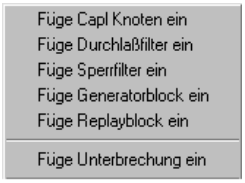
Für Trace-Fenster und das Write-Fenster lassen sich mit der Funktion *Font|Wählen* im Kontextmenu der Font des jeweiligen Fensters einstellen. Im Meßaufbau sind zusätzlich die Fonts der Funktionsblöcke und die Fonts der Kommentare zu den Funktionsblöcken einstellbar.

Mit der Funktion *Font|Zurücksetzen* wird der für das entsprechende Fenster vordefinierte Font gesetzt.

2.6 Hot-Spots und einfügbare Funktionsblöcke

Im Datenflußdiagramm sind zwischen den Grundfunktionsblöcken quadratische Punkte (Hot-Spots) vorhanden, an denen weitere Funktionsblöcke eingefügt oder der Datenfluß gesperrt werden kann. Die Hot-Spots selbst lassen alle Daten ungehindert passieren.

Wird ein Hot-Spot mit der rechten Maustaste angeklickt (oder mit den Cursortasten ausgewählt und anschließend mit <F10> aktiviert) so erscheint ein Kontextmenü mit folgenden Menüpunkten:



Beim Anklicken eines der ersten fünf Menüpunkte wird ein Block eingefügt, der die entsprechende Funktion erfüllt (die dazugehörigen Funktionsblöcke werden in den folgenden Unterkapiteln beschrieben). Vor und nach diesem Block erscheinen neue Hot-Spots, so daß weitere Blöcke eingefügt werden können. Eine Besonderheit bildet der letzte Menüpunkt. Bei seiner Aktivierung erscheint ein unterbrochener Hot-Spot, der damit darstellt, daß der Informationsfluß an dieser Stelle gesperrt ist.

Funktionsblöcke lassen sich im Datenflußbild an ihren Aufschriften erkennen. Ein „P“ steht hierbei für CAPL-Knoten (*P*rogrammblock), „DF“ und „SF“ bezeichnen Durchlaß- und Sperrfilter, ein „G“ weist auf einen Generatorblock hin und „R“ steht schließlich für einen Replay-Block.

Alle fünf Blöcke besitzen Kontextmenüs, die man entweder durch Klicken mit der rechten Maustaste oder durch Auswählen des Blockes im Datenflußplan und anschließendem Drücken von <F10> öffnen kann. Der erste Menüeintrag öffnet bei jedem Block einen Konfigurationsdialog, der zur Parametrierung des Blocks dient (Beim Generatorblock existieren zwei Konfigurationsdialoge, die über die ersten beiden Menüeinträge geöffnet werden können). Durch Anwählen des letzten Eintrags im Kontextmenü *Lösche diesen Knoten* können Sie den Block wieder aus dem Datenflußplan entfernen. Dabei gehen alle Konfigurationsinformationen verloren. Die CAPL-Quelldateien des CAPL-Knotens und die Logging-Datei des Replay-Blocks werden allerdings nicht gelöscht.

Über das Kontextmenü können Sie jedem Funktions- und Analyseblock einen Kommentar zuordnen. Standardmäßig ist die Kommentaranzeige aktiviert. Mit der Funktion *Kommentar anzeigen* im Kontextmenü können Sie die Anzeige für jeden Block getrennt ein- und ausschalten.

Beachten Sie, daß der Umfang des angezeigten Kommentares von der aktuellen Position des Funktionsblockes und dem damit für die Anzeige vorhandenen Platzes abhängig ist. Unabhängig hiervon werden immer nur die ersten beiden Zeilen des Kommentares angezeigt. Es empfielt sich für die Anzeige in den ersten beiden Zeilen nur kurze Stichworte anzugeben und weiter unten im Text umfangreichere Informationen einzutragen.

Sollten Sie keinen Kommentar angegeben haben, die Anzeige jedoch angeschaltet ist, so wird ein voreingestellter Kommentar angezeigt.

2.6.1 CAPL-Knoten

Ein CAPL-Knoten ist ein universeller Funktionsblock, dessen Eigenschaften der Anwender durch das Schreiben eines CAPL-Programms bestimmt. Eine detaillierte Beschreibung von CAPL-Programmblöcken und Hinweise zum Erstellen von CAPL-Programmen finden Sie in Kapitel 3.

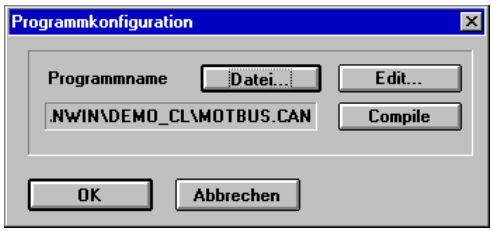


Abbildung 52: Konfigurationsdialog für CAPL Programme

Eine wichtige Anwendung von Programmblöcken ist z.B. die Vorgabe von Sendebotschaften im Sendezweig. Es können jedoch auch sehr einfach Programmblöcke zur Datenreduktion (Beispiel: „Nur jede zehnte Botschaft durchlassen“) oder zur Überwachung geschrieben werden.

Hinweis: Ein CAPL-Knoten sperrt in einem Datenflußzweig alle Botschaften, die im Programm nicht explizit mit **`output()`** ausgegeben werden. Ein Programm, das für alle Botschaften durchlässig ist, muß daher folgende Message-Prozedur enthalten:

```
on message * {
   output(this);  /* alle Botschaften durchlassen */
}
```

Durch gezieltes Einsetzen von **`output(this)`**, lassen sich also auch mit CAPL-Programmen im Auswertezweig Filter programmieren, deren Funktionalität sehr viel komplexer sein kann als die von normalen Durchlaß- oder Sperrfiltern.

Programmblöcke erscheinen im Datenflußplan als kleine Blöcke mit der Aufschrift *P*.

Ordnen Sie im Konfigurationsdialog dem Programmblock zunächst einen CAPL-Dateinamen (Erweiterung `*.CAN`) zu.

Drücken Sie *Edit* um den CAPL-Browser zu öffnen. Der Browser ist ein übersichtliches Werkzeug zur Erstellung, Modifikation und Compilierung von CAPL-Programmen und wird ausführlich in Abschnitt 3.4 beschrieben.

Bevor Sie die Messung starten, müssen Sie die CAPL-Datei compilieren. Drücken Sie dazu die Schaltfläche *Compile* oder wählen Sie im Hauptmenü *Konfiguration|Compiliere alle Knoten*, um alle CAPL-Programme auf einmal zu compilieren.

Hinweis: Es ist zulässig, gleiche CAPL-Programme in verschiedenen Programmblöcken zu referenzieren. Dies ist z.B. interessant, wenn in zwei unterschiedlichen Datenflußzweigen gleiche Datenmanipulationen vorgenommen werden sollen (z.B. Datenreduktionen).

Durch Betätigen der Leertaste oder durch Anwählen der Zeile *Knoten aktiv* im Kontextmenü des CAPL-Knoten deaktivieren Sie den Knoten; nach erneutem Ausführen der gleichen Aktion wird er wieder aktiv. Der Menüpunkt *Lösche diesen Knoten* entfernt den CAPL-Knoten aus dem Meßaufbau.

Hinweis: Beim Entfernen eines CAPL-Knotens aus dem Meßaufbau wird die CAPL-Quelldatei nicht gelöscht.

2.6.2 Durchlaßfilter

Mit Durchlaßfiltern kann eine gezielte Reduktion der Datenmenge vorgenommen werden. Es werden nur die angegebenen Identifier und/oder Identifier-Bereiche durchgelassen. Zusätzlich kann die Übertragung auf bestimmte Botschaftsarten beschränkt werden. Außerdem kann eingestellt werden, ob sich die Filterung auch auf Error-Frames beziehen soll. Overload-Frames werden hierbei als Error-Frames behandelt.

Durchlaßfilter erscheinen im Datenflußplan als kleine Blöcke mit der Aufschrift *DF*. Beim Doppelklicken auf diese Blöcke erscheint ein Konfigurationsdialog mit dem der Filter parametriert werden kann. Dabei können Identifier(bereiche) und Attribute eingegeben werden.

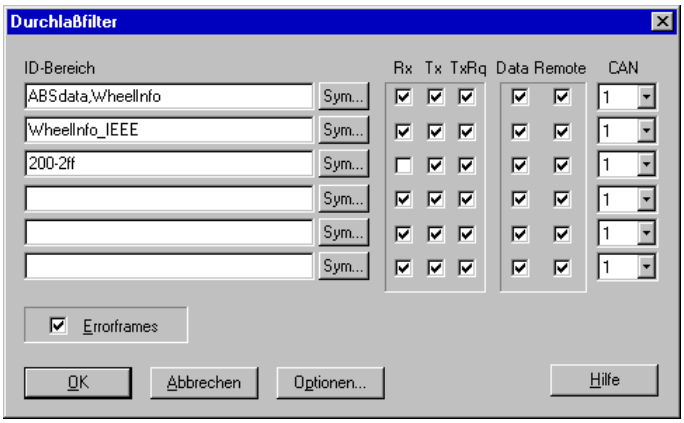


Abbildung 53: Filterkonfigurationsdialog

Beim Entfernen eines Durchlaßfilters aus dem Meßaufbau (Anwählen des Eintrags *Konfiguration löschen* im Kontextmenü) geht die Filterkonfiguration verloren.

Hinweis: Ein Durchlaßfilter, das nicht konfiguriert (leer) ist, läßt sinngemäß keine Botschaften durch und sperrt somit den gesamten Botschaftsverkehr.

2.6.3 Sperrfilter

Mit Sperrfiltern können Sie gezielt bestimmte Identifier und/oder Identifier-Bereiche aus dem Datenstrom ausblenden. Die Parametrierung erfolgt so wie im Durchlaßfilter. Sperrfilter erscheinen im Datenflußplan als kleine Blöcke mit der Aufschrift *SF*.

2.6.4 Replay-Block

Der Replay-Block bietet Ihnen die Möglichkeit, bereits aufgezeichnete Meßabläufe wiederzugeben. Die wichtigste Anwendung ist die Wiedergabe eines aufgezeichneten Datenstroms auf den CAN-Bus.

Replay-Blöcke erscheinen im Datenflußplan als kleine Blöcke mit der Aufschrift *R*. Geben Sie im Konfigurationsdialog die Logging-Datei an, die wiedergegeben werden soll. Zusätzlich können Sie spezifizieren, wie gesendet werden soll. Dies beinhaltet eine Auswahl der Botschaften und das zeitliche Verhalten:

Sie können angeben, ob Rx-Botschaften bzw. Tx-Botschaften gesendet werden sollen oder nicht. Sie können ferner angeben, ob die vom CAN-Controller 1 stammenden Botschaften auf CAN 1 oder CAN 2 oder gar nicht gesendet werden sollen und ebenso für vom CAN 2 stammende Botschaften.

Die Datei kann einmalig oder periodisch gesendet werden. Bei periodischem Senden wird nach Ende der Datei wieder mit ihrer ersten Botschaft begonnen.

Der *Sendezeitpunkt* bestimmt, wann die erste Botschaft der Datei gesendet werden soll. Es gibt hierbei drei Möglichkeiten:

Sofort	Die erste Botschaft wird bei Start der Messung gesendet.
Original	Der Sendezeitpunkt wird durch die Zeit, die mit der Botschaft in der Datei gespeichert ist, bestimmt.
Vorgabe	Sie können den Zeitpunkt, zu dem die erste Botschaft gesendet wird, explizit in Millisekunden ab Start der Messung vorgeben.

In allen drei Fällen bleibt der zeitliche Abstand zwischen den Botschaften innerhalb der Datei erhalten. Ist er geringer als eine Millisekunde, wird die Sendung entsprechend verzögert.

Beim Entfernen eines Replay-Blocks aus dem Meßaufbau (Anwählen des Eintrags *Konfiguration löschen* im Kontextmenü) geht die Konfiguration verloren. Dies betrifft nur die Einstellungen der Dialogbox. Die Replay-Datei selber wird nicht gelöscht.

Hinweis: Damit ein Replay-Block Daten auf den Bus senden kann, müssen Sie ihn im Sendezweig des CANalyzers einfügen. Falls Sie ihn in den Auswertezweig einfügen, werden die Daten zwar in die rechts gelegenen Auswerteblöcke gesendet und in den jeweiligen Fenstern angezeigt. Sie gelangen aber wegen des von links nach rechts gerichteten Datenflußes nicht auf den Bus.

2.6.5 Generatorblock

Der Generatorblock dient der Erzeugung von Botschaften zum Senden. Hierzu werden Auslösebedingungen und eine Sendeliste definiert. Als Auslösebedingung kann eine empfangene Botschaft, ein Tastendruck oder eine Periodendauer angegeben werden. Die Auslösebedingungen können auch kombiniert verwendet werden. In der Sendeliste werden Botschaften und Error-Frames in der Reihenfolge eingetragen, wie sie gesendet werden sollen.

Jeweils beim Eintreten einer Auslösebedingung wird die nächste Botschaft gesendet. Beim Erreichen des Listenendes wird je nach Konfiguration wieder am Listenanfang begonnen.

Da sich Generatorblöcke sehr aufwendig konfigurieren lassen, bietet Ihnen das Kontextmenü, die Möglichkeit einen Generatorblock als Datei abzuspeichern und später - auch aus einer anderen Konfiguration heraus - wieder zu laden. Sie können Gereratorblöcke also problemlos auch zwischen verschiedenen Konfigurationen austauschen.

Generatorblöcke erscheinen im Datenflußplan als kleine Blöcke mit der Aufschrift *G*.

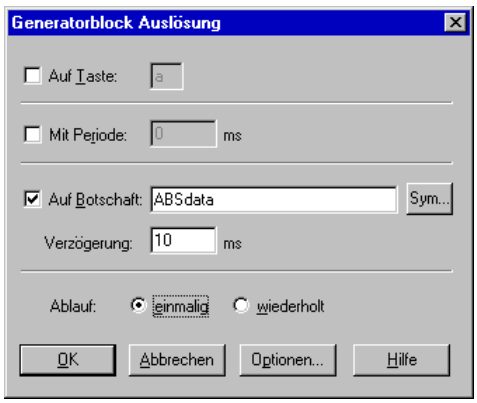


Abbildung 54: Generatorblockkonfiguration-Auslösung

Hinweis: Damit ein Generatorblock Daten auf den Bus senden kann, müssen Sie ihn im Sendezweig des CANalyzers einfügen. Falls Sie ihn in den Auswertezweig einfügen, werden die Daten zwar in die rechts gelegenen Auswerteblöcke gesendet und in den jeweiligen Fenstern angezeigt. Sie gelangen aber wegen des von links nach rechts gerichteten Datenflußes nicht auf den Bus.

2.6.5.1 Konfiguration der Auslösung

Wählen Sie im Kontextmenü den Eintrag *Auslösung*, um eine Auslösebedingung für das Senden von Botschaften zu definieren.

Sie können eine oder mehrere der drei möglichen Bedingungen auswählen, indem Sie die jeweilige Checkbox ankreuzen wird. Bei *Auf Botschaft* geben Sie an, welcher Identifier das Senden auslösen soll und mit welcher Verzögerung es erfolgen soll, bei *Auf Taste* geben Sie die auslösende Taste an, und bei *Auf Periode* die Periodendauer in Millisekunden.

In der Zeile *Ablauf* stellen Sie ein, ob die Sendeliste genau einmal oder periodisch abgearbeitet werden soll.

2.6.5.2 Konfiguration der Sendeliste

Wählen Sie im Kontextmenü den Eintrag *Sendeliste* oder doppelklicken Sie auf den Generatorblock um die Liste der zu sendenden Botschaften zu erstellen:

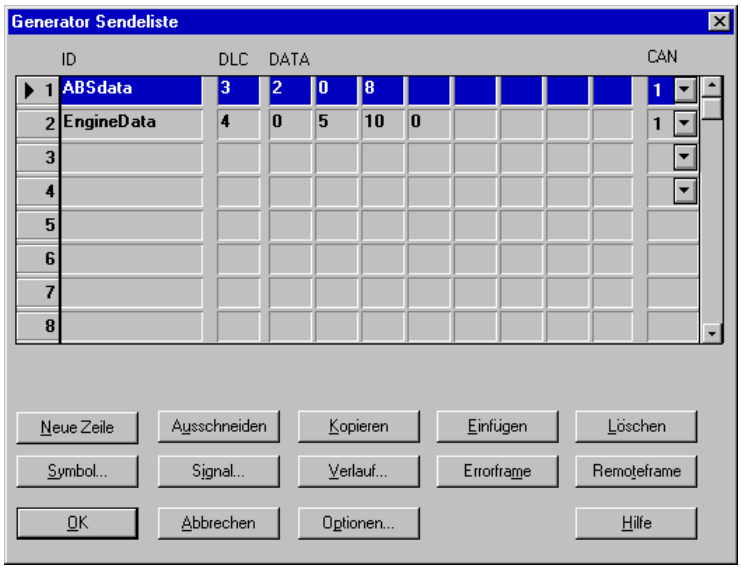


Abbildung 55: Generatorblockkonfiguration-Sendeliste

Beim Eintreten einer der Triggerbedingungen wird das jeweils nächste Element dieser Liste gesendet. Ist das Ende der Liste erreicht, wird wieder mit dem ersten Element begonnen, wenn der Ablaufmodus *periodisch* eingestellt ist. Die Liste kann auch aus nur einem Element bestehen.

In der Dialogbox werden neun Zeilen dargestellt. Die *aktuelle Zeile,* auf die sich die Buttons des Dialogs beziehen, ist durch das Zeichen '»' am Zeilenanfang gekennzeichnet. Die aktuelle Zeile wird durch Betätigen der <TAB>-Taste sowie durch Klicken mit der Maus auf die Eingabefelder des Dialogs automatisch mitbewegt.

Jede Zeile der Liste besteht aus 11 Spalten. In der ersten Spalte geben Sie den gewünschten Identifier an. Das Feld DLC bestimmt die Anzahl der Datenbytes der Botschaft. Dahinter kommen die Felder für die Datenbytes. Es werden nur so viele übernommen, wie bei DLC angegeben ist. Der Rest wird ignoriert. Die letzte Spalte ist eine Combobox zur Auswahl des Controllers, über den die Botschaft gesendet werden soll.

2.6.5.3 Eingabe von Signalwerten

Sofern Sie mit einer Datenbasis arbeiten, können Sie den Generatorblock symbolisch konfigurieren. Verwenden Sie die Optionsschaltfläche *Symbol...*, um aus

der Datenbasis eine Botschaft auszuwählen, die vom Generatorblock gesendet werden soll.

Mit der Schaltfläche *Signal...* können dann die physikalischen Signalwerte der Botschaft definiert werden.

Im Signalwert-Eingabedialog werden die einzelnen Signale der Botschaft dargestellt, die in der Sendeliste des Generatorblocks als aktuelle Zeile aktiv war. Falls die Botschaft nur Standardsignale enthält, besteht die Signalliste aus drei Spalten. In den Zeilen sind die einzelnen Signale aufgeführt. In der letzten Spalte können Sie einen Signalwert vorgeben. Falls Sie dem Signal in der Datenbasis eine Wertetabelle zugeordnet haben, können Sie statt eines numerischen Wertes auch die entsprechende symbolische Beschreibung zuordnen. Wählen Sie diese in der mittleren Spalte aus der Signalwertetabelle aus.

Die Speicherung der physikalischen Größen erfolgt in den CAN-Botschaften in diskreter Form. Daher ist der angegebene Zahlenwert im Feld *Wert* nicht immer auf einen diskreten Wert abbildbar. In diesem Fall werden beim Verlassen der Zeile oder Aktivieren des *OK*-Buttons die beiden nächstmöglichen physikalischen Werte in einem Dialog angezeigt. Anschließend wird der eingegebene Wert auf den nächstliegenden möglichen Wert gerundet.

Beispiel:

Das Signal *Motordaten.Drehzahl* sei definiert als 16 Bit unsigned mit einem Offset von 0 und einem Faktor von 10. Wird als zu vergleichender Wert 1015 angegeben, müßte der Rohwert 1015/10 = 101.5 sein. Da nur diskrete Werte vorkommen, muß entweder 101 oder 102 verwendet werden, was physikalischen Größen von 1010 oder 1020 entspricht. Genau diese beiden Zahlen erscheinen in den Dialog.

Trotz diskreter Speicherung können gültige Größen Nachkommastellen haben. Wird im obigen Beispiel mit einem Faktor von 10.5 gerechnet, so werden als mögliche Werte 1008 und 1018.5 vorgeschlagen.

2.6.5.4 Eingabe modusabhängiger Signale

In der Regel besteht eine direkte, feste Zuordnung eines Datensegments einer Botschaft zu einem Signal. Bei Multiplexbotschaften werden jedoch in den Datensegmenten in Abhängigkeit von einem Modussignal unterschiedliche Signale übertragen. Für jeweils einen Moduswert ist dann immer nur ein Teil aller möglichen Signale der Botschaft definiert.

Im Dialog zur Eingabe der Signalwerte werden zur einfacheren Eingabe die Standardsignale und modusabhängigen Signale in zwei unabhängigen Listboxen dargestellt. Zusätzlich werden in der modusabhängigen Listbox alle Signale, die im aktuellen Modus nicht definiert sind herausgefiltert.

Das *Modussignal* und der *Modus* werden in den zugehörigen Edit-Feldern angezeigt. Wenn der Moduswert geändert, wird, wird die Liste mit modusabhängigen Signalen neu aufgebaut. Das Modussignal selbst kann hier nicht geändert werden.

2.6.5.5 Funktionsgenerator für die Sendeliste

Oft möchte man nicht nur bestimmte Signalwerte auf den Bus legen, sondern ganze Signalverläufe vorgeben. Zu diesem Zweck besitzt der Generatorblock einen Signalverlaufsgenerator, den Sie mit der Schaltfläche *Verlauf...* konfigurieren.

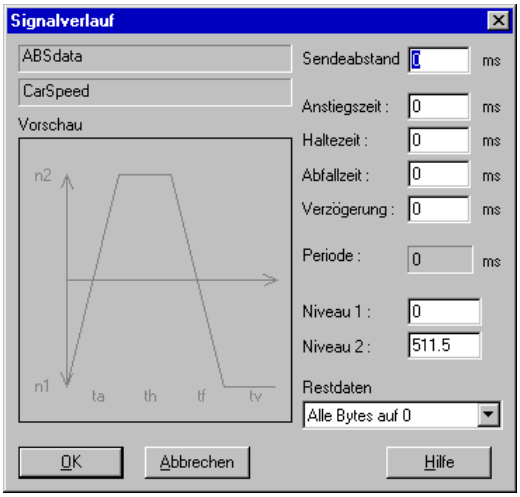


Abbildung 56: Signalverlaufsgenerator

Der Konfigurationsdialog gestattet die Parametrierung einer Trapezfunktion. Beim Verlassen des Dialogs mit *OK* werden die entsprechenden Zeilen in der Generatorblock-Sendeliste automatisch generiert. Folgende Signalverläufe lassen sich mit bestimmten Parametern erzeugen:

Signalverlauf	Parameter
Rechteck	ta = tf = 0
Dreieck	th = tv = 0
Sägezahn	th = tf = tv = 0 oder
	ta = th = tv = 0
Konstante	n1 = n2

Im oberen Teil der Dialogbox wird die Botschaft und das Signal angezeigt. Darunter, im Vorschaufeld ist ein Trapez und die Bedeutung seiner Parameter dargestellt. Durch Nullsetzen einzelner Parameter lassen sich auch folgende Verläufe erzeugen:

Die Niveaus n1 und n2 müssen physikalisch angegeben werden. Hier gelten die entsprechenden Einschränkungen (siehe z.B. Generatorblock Signalwerte).

Die Angabe Sendeabstand bezeichnet den Abstand zwischen je zweier Botschaften und entspricht der Angabe im Dialog Generatorblock *Triggerauslösung Periode*. Da CAN botschafts- und nicht signalorientiert ist, erhalten alle Signale einer Botschaft den gleichen Sendeabstand!

Da es sich bei diesem Signalverlaufsgenerator nur um ein Hilfsmittel für den Generatorblock und nicht um einen eigenständigen Block handelt, ist bei der Verwendung unterschiedlicher Periodendauern für mehrere Signale innerhalb einer Botschaft folgendes zu beachten:

Der Generator erzeugt eine Liste von Botschaften mit

Anzahl Botschaften = Periode / Sendeabstand,

also genau eine Periode. Enthält die Sendeliste bei Aufruf von *Verlauf* mehr Botschaften, werden diese verworfen. Über die Combobox *Restdaten* kann festgelegt werden, wie die restlichen Signale bei zusätzlich erzeugten Botschaften behandelt werden:

Alle Byte auf 0:

Alle Signale werden auf den Rohwert 0 gesetzt.

Periodisch fortsetzen:

Beim Erzeugen werden die bisherigen Botschaften so oft kopiert, bis die neue Periodenlänge erreicht ist.

Beispiel: Die Originalliste enthält 3 Botschaften, die neue Periodenlänge erfordert 9 Botschaften. Das neue Signal (Byte 1) ist ein Sägezahn.

Original-Sendeliste:	Neue Sendeliste:
0 0 0 4 0	1 0 0 4 0
0 1 0 2 0	2 1 0 2 0
0 2 0 4 0	3 2 0 4 0
	4 0 0 4 0
	5 1 0 2 0
	6 2 0 4 0
	7 0 0 4 0
	8 1 0 2 0
	9 2 0 4 0

Die empfohlene Vorgehensweise bei Generierung mehrerer Signale ist daher folgende:

Die Signale werden beginnend mit der kürzesten Periodendauer festgelegt. Es werden immer ganzzahlige Vielfache der Periodendauer verwendet.

2.6.6 Unterbrechung

Sollen bestimmte Zweige des Datenflusses nicht durchlaufen werden, kann aus einem Hot-Spot ein Unterbrechungspunkt erzeugt werden. Dies ist z.B. im Online-Modus sinnvoll, wenn in Folge einer hohen Datenrate nicht mehr alle Funktionen (vor allem das Trace-Fenster) ohne Datenverlust bedient werden können.

Beim Einbringen einer Unterbrechung bleibt die Konfiguration hinter der Unterbrechung voll erhalten, so daß nach dem Löschen der Unterbrechung wieder der alte Zustand vorliegt. Die Unterbrechung bietet somit eine sehr schnelle Möglichkeit bestimmte Datenpfade temporär abzukoppeln und damit Rechenzeit einzusparen.

Beim Start einer Messung wird der momentan gültige Datenflußplan in einen internen Strukturbaum umgesetzt. Bei diesem Umsetzvorgang werden Unterbrechungen, die in einem Pfad entdeckt werden, nach vorne bis zum nächsten Verzweigungspunkt propagiert. Deshalb ist es für die Prozessorbelastung während der Messung gleichgültig, ob eine Unterbrechung vorne oder hinten in einem Pfad gesetzt wird. Es wird immer der gesamte Pfad ausgeblendet.

2.7 Aufzeichnen und Auswerten von Meßdateien

Der CANalyzer bietet Ihnen die Möglichkeit den CAN-Datenverkehr in eine Logging-Datei abzuspeichern, um diese später im Offline-Modus auszuwerten.

Zu diesem Zweck stehen Ihnen Logging-Blöcke zur Verfügung. Aufgabe eines Logging-Blockes ist es, die an seinem Eingang anstehenden Daten in eine Datei abzulegen. Sie konfigurieren die Logging-Datei im Meßaufbau über das Dateitrommelsymbol ganz rechts im Logging-Zweig.

Um die Daten möglichst schon bei der Erfassung so weit wie möglich zu reduzieren, ist jeder Logging-Block mit einer komfortablen Triggerung ausgestattet. Diese gestattet es, eine Triggerbedingung zu formulieren und dann nur das Umfeld des Trigger-Zeitpunkts abzuspeichern. Während jeder Messung können mehrere Trigger auf unterschiedliche Ereignisse ausgelöst werden, für die Vor- und Nachlaufzeiten angegeben werden. Die Triggerbedingung läßt sich frei programmieren. Sie konfigurieren die Triggerung im Meßaufbau über den Funktionsblock *Logging*.

Um Logging-Dateien zu analysieren, verfügt der CANalyzer über einen Offline-Modus. Im Gegensatz zum Online-Modus dient hier als Datenquelle eine Datei, wie sie z.B. im Online-Modus durch das Logging erzeugt wurde. Alle Meß- und Auswertefunktionen des Online-Modus stehen Ihnen auch im Offline-Modus zur Verfügung.

2.7.1 Logging-Trigger

Durch doppeltes Klicken auf den Logging-Block im Datenflußplan oder durch Auswählen des Logging-Blocks mit den Cursortasten und anschließendes Betätigen der Eingabetaste öffnen Sie den Konfigurationsdialog des Logging-Triggers.

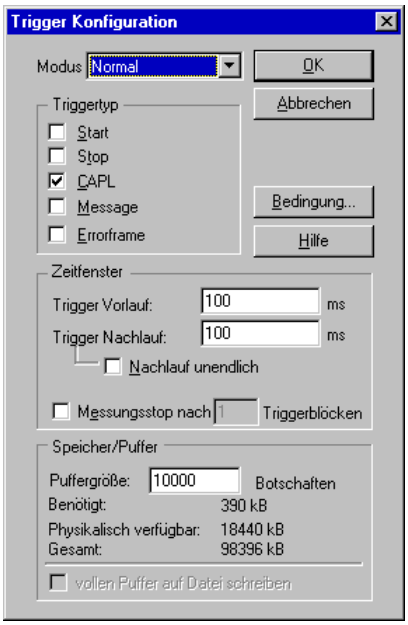


Abbildung 57: Trigger Konfigurationsdialog

2.7.1.1 Triggermodi

Sie können Auslösemechanismus für das Logging in drei Modi betreiben:

Im Modus *Gesamte Messung* wird die gesamte Messung aufgezeichnet. Vor- und Nachlaufzeiten sind nicht wählbar. Mit der Schaltfläche *Vollen Puffer auf Datei schreiben* wird ein voller Datenpuffer bei langen Messungen immer zwischengesichert. Sie können so einen Datenverlust vermeiden.

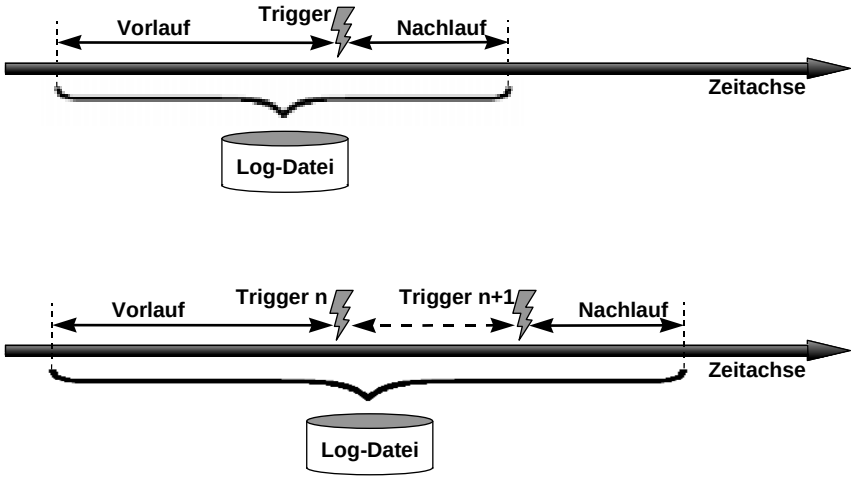


Abbildung 58: Zeitfenster für den Trigger im Normal-Modus (oben) und im Toggle-Trigger-Modus (unten)

Im *Normal-Modus* wird das benutzerdefinierbare Zeitfenster zwischen Vor- und Nachlaufzeit um jeden Trigger gelegt. Vor- und Nachlaufzeit beziehen sich also auf denselben Trigger. Alle Daten, die in dieses Zeitfenster fallen, werden in der Logging-Datei aufgezeichnet.

Im *Toggle-Trigger-Modus* wird das Zeitfenster durch jeweils zwei aufeinanderfolgende Trigger (Blockanfang- und Blockende-Trigger) beschrieben. Der erste während der Messung ausgelöste Trigger ist ein Blockanfang-Trigger, der zweite ein Blockende-Trigger. Danach folgt wieder ein Blockanfang-Trigger usw. Die Vorlaufzeit bezieht sich im Toggle-Trigger-Modus auf einen Blockanfang-Trigger, die Nachlaufzeit auf einen Blockende-Trigger.

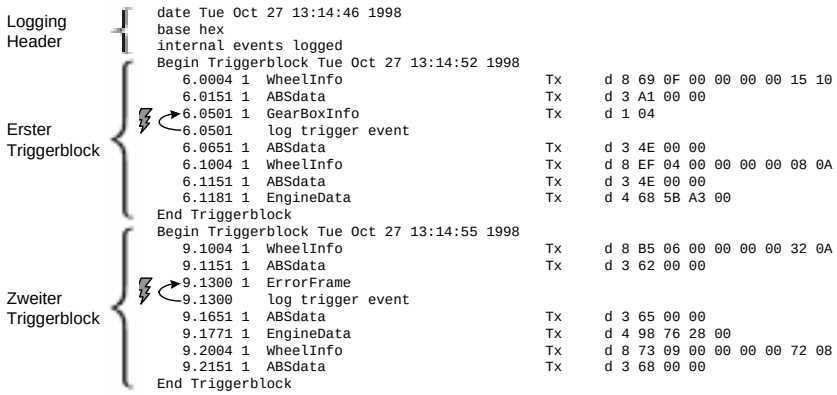

Logging Header
Erster Triggerblock
Zweiter Triggerblock
date Tue Oct 27 13:14:46 1998
base hex
internal events logged
Begin Triggerblock Tue Oct 27 13:14:52 1998
6.0004 1 WheelInfo Tx d 8 69 0F 00 00 00 00 15 10
6.0151 1 ABSdata Tx d 3 A1 00 00
6.0501 1 GearBoxInfo Tx d 1 04
6.0501 log trigger event
6.0651 1 ABSdata Tx d 3 4E 00 00
6.1004 1 WheelInfo Tx d 8 EF 04 00 00 00 00 08 0A
6.1151 1 ABSdata Tx d 3 4E 00 00
6.1181 1 EngineData Tx d 4 68 5B A3 00
End Triggerblock
Begin Triggerblock Tue Oct 27 13:14:55 1998
9.1004 1 WheelInfo Tx d 8 B5 06 00 00 00 00 32 0A
9.1151 1 ABSdata Tx d 3 62 00 00
9.1300 1 ErrorFrame
9.1300 log trigger event
9.1651 1 ABSdata Tx d 3 65 00 00
9.1771 1 EngineData Tx d 4 98 76 28 00
9.2004 1 WheelInfo Tx d 8 73 09 00 00 00 00 72 08
9.2151 1 ABSdata Tx d 3 68 00 00
End Triggerblock

Abbildung 59: Logging-Datei mit 2 Triggerblöcken. Vorlauf: 50ms, Nachlauf: 100ms, Triggertypen: Botschaft *GearBoxInfo* und Error-Frames

Beispielsweise läßt sich im Toggle-Trigger-Modus die gesamte Messung aufzeichnen, indem man als Triggerbedingungen Start und Stop auswählt. Bei Messungsstart wird dann ein Blockanfang-Trigger ausgelöst. Die Messung wird bis zum Auftreten des Blockende-Triggers bei Messungsstopp mitgeloggt. Vor- und Nachlaufzeiten werden bei dieser Einstellung ignoriert.

Sie können bestimmen, ob der Triggervorgang ein- oder mehrmals ablaufen soll. Ist die Option *Messungsstopp nach n Triggern* im Konfigurationsdialog des Triggers ausgewählt, so wird nach Ablauf der Nachlaufzeit des n-ten Triggers der Meßvorgang vom Logging-Block gestoppt

2.7.1.2 Trigger-Ereignisse

Mit den fünf Checkboxen im Konfigurationsdialog können Sie die Trigger-Ereignisse festlegen. Folgende Typen stehen zur Verfügung:

- Triggerung beim Start. Die Triggerung erfolgt sofort beim Start der Messung. Der Aufzeichnungszeitraum wird hier allein durch die Nachlaufzeit bestimmt.
- Triggerung bei Stop. Die Triggerung erfolgt bei Ende der Messung. Dies kann durch die CAPL-Funktion **`stop()`** oder durch die Taste <Esc> ausgelöst werden. Der Aufzeichnungszeitraum wird hier allein durch die Vorlaufzeit bestimmt.
- Triggerung beim Auftreten bestimmter Botschaften. Dazu können in Dialogboxen Attribute ausgewählt werden und Masken für ID und Datenbyte etc. angegeben werden. Eine Beschreibung dieser Triggerbedingungen finden Sie in Abschnitt 2.7.5. Beim Eintreffen der dort spezifizierten Bedingungen wird die Triggerung ausgelöst.
- Triggerung über ein CAPL-Programm: Mit den Methoden der *CAPL-Programmierung* lassen sich beliebig komplexe Bedingungen formulieren, die vom Eintreffen verschiedener Ereignisse abhängig sein können. Sobald über eine

Prozedur die CAPL-Funktion **`trigger()`** aufgerufen wird, wird die Triggerung ausgelöst.

- Triggerung auf Error-Frames. Die Triggerung erfolgt bei jedem Auftreten eines Error-Frames.

2.7.1.3 Zeitfenster

Im Konfigurationsdialog legen Sie auch das Zeitfenster (die Vor- und die Nachlaufzeit in Millisekunden) um jeden Trigger fest.

Die Vorlaufzeit bestimmt, welcher Zeitabschnitt vor dem Zünden der Triggerbedingung gespeichert werden soll. Zündet die Triggerbedingung so früh, daß seit dem Start der Messung die Vorlaufzeit noch nicht verstrichen ist, so können nur die bisher angefallenen Daten gespeichert werden.

Die Nachlaufzeit gibt an, wie lange noch Daten nach dem Zünden der Triggerbedingung erfaßt und gespeichert werden. Sie können festlegen, ob die Messung nach Ablauf der Nachlaufzeit automatisch beendet werden soll.

2.7.1.4 Konfigurieren des Logging-Puffers

Der CANalyzer speichert die im Logging-Zweig eintreffenden Daten zunächst in einem Ringpuffer im Hauptspeicher des PCs.

In der letzten Zeile des Dialogs legen Sie die Größe dieses Puffers fest. Pro Botschaft werden etwa 50 Byte Speicher belegt, die allerdings erst während der Messung bei Bedarf allokiert werden. Bei sehr großen Puffergrößen kann es daher vorkommen, daß Windows während der Messung Teile des Hauptspeichers auf die Festplatte auslagert, so daß der Programmablauf u.U. erheblich verzögert wird.

Nach dem Eintritt einer Triggerbedingung wartet das Logging, bis die Nachlaufzeit abgelaufen ist, während weiterhin der Ringpuffer mit Rohdaten gefüllt wird. Nach dem Ablaufen der Nachlaufzeit werden die Daten vom Ringpuffer in die eigentliche Logging-Datei übertragen. Abhängig von der Datenmenge kann dieser Umspeichervorgang mit merklichen Wartezeiten verbunden sein.

Da der Ringpuffer nur eine begrenzte Größe hat, kann es vorkommen, daß bei zu langen Vor- bzw. Nachlaufzeiten Daten verlorengehen. Es werden dann nur die letzten Daten bis zum Ablauf der Nachlaufzeit aufgezeichnet und es wird eine Fehlermeldung ausgegeben.

Hinweis: Wenn Sie bei einer Langzeitmessung alle Daten auf Datei schreiben wollen, wählen Sie die Option *Vollen Puffer auf Datei schreiben*. Die Option bewirkt, daß während der Messung ein voller Puffer auf Datei geschrieben wird. Bei dieser Option ist die Puffergröße entscheidend. Damit die Systembelastung beim Abspeichern des Puffers nicht zu hoch ist, und es nicht zu Datenverlusten kommt, sollte der Puffer nicht allzu groß eingestellt werden. In der Praxis haben sich Puffer für ca. 1000 Botschaften bewährt. Der Knopf ist nur im Modus *Gesamte-Messung* oder im Normal-Modus mit Triggertyp *Start* verfügbar.

Um auch bei hoher Systembelastung Datenverluste zu vermeiden, sollten Sie folgende Punkte beachten:

- Schließen Sie alle Anwendungen, die im Hintergrund ablaufen und Systemzeit beanspruchen,
- Schalten Sie alle Blöcke im Meßaufbau auf *zyklischer Update* oder u.U auch ganz ab.
- Setzen Sie Filter zur Datenreduktion ein.
- Führen Sie während der Messung keine unnötigen Benutzeraktionen, wie z.B. das Verschieben von Fenstern durch.

Sollte trotzdem ein Datenverlust auftreten, so können Sie diesen mit der Option *Data Lost Messagebox* im Logging-Dialog sichtbar gemacht werden. Diese Option hat zur Folge, daß bei Datenverlust nach Messungsende eine Dialogbox angezeigt wird. Erscheint keine Meldung, wissen Sie, daß alle Daten aufgezeichnet wurden.

In der Logging-Datei werden mit Hilfe dieser Option die korrupten Stellen mit einem '*' als Sonderzeichen markiert. Das Zeichen für den Datenverlust verschwindet erst, wenn der Überlastbetrieb beendet ist.

2.7.2 Logging-Dateien

Über das Kontextmenü der Datentrommel rechts vom Triggerblock öffnen Sie den Konfigurationsdialog für die Logging-Datei.

Abbildung 60: Logging Konfiguration

Es stehen Ihnen 2 Formate zur Verfügung in denen Sie Logging-Dateien abspeichern können: *binär* und *ASCII*. Bei längeren Online-Messungen ist das

Binärformat u.U. sinnvoller, da dieses um einen Faktor 2 bis 3 kleinere Logging-Dateien liefert. Bei Anwahl des ASCII-Formate werden die Daten als lesbare Textdatei gespeichert. Der Zeilenaufbau entspricht der Ausgabe im Trace-Fenster. Dezimal- oder Hexadezimaldarstellung wird durch die Einstellung des globalen Schalters bestimmt. Die ASCII-Formate dienen auch zum Datenaustausch mit Fremdprogrammen oder zur Übernahme von Trace-Daten in Dokumente.

Die Datenquelle des Offline-Modus kann sowohl mit Binär- als auch mit ASCII-Dateien konfiguriert werden.

Die automatisch vorgegebene Erweiterung des Dateinamens ist `.LOG` für eine binäre Logging-Datei bzw. `.ASC` für eine ASCII-Datei. Als Default-Name wird `CANWIN.LOG` bzw. `CANWIN.ASC` vorgeschlagen.

Um eine ASCII-Datei einzusehen bzw. zu editieren, dopplklicken Sie die Dateitrommel oder betätigen Sie die Schaltfläche *Datei editieren* im Konfigurationsdialog der Logging-Datei. Sie können hierbei ihren eigenen Texteditor verwenden. Tragen Sie dazu in der Datei `CAN.INI` im Abschnitt `[Environment]` die Zeile

```
LogEditor=MYEDITOR.EXE
```

ein, wobei Sie für `MYEDITOR.EXE` den Namen Ihres eigenen Editors angeben müssen.

Sie können das Logging so konfigurieren, daß der gewählte Logging-Dateiname nach Ende der Messung bzw. am Ende eines Triggerblocks automatisch inkrementiert wird. Dadurch vermeiden Sie, daß bereits bestehenden Dateien überschrieben werden

Beispiel:

Logging-Datei vor der Messung: `C:\CANWIN\TEST_CL\MESS_01.ASC`

Logging-Datei nach der Messung: `C:\CANWIN\TEST_CL\MESS_02.ASC`

Im Konfigurationsdialog können Sie angeben, ob Ihnen Datenverluste in Überlastsituationen (vgl. Abschnitt 4.2) mitgeteilt werden sollen. In der Logging-Datei werden dabei die fehlerhaften Zeilen mit einem '*' als Sonderzeichen markiert.

Hinweis: Analog zur Ausgabe ins Write-Fenster mit der CAPL-Funktion **`write()`**, können Sie mit den Funktionen **`writeToLog()`** und **`writeToLogEx()`** aus CAPL-Programmen Textzeilen in ASCII-Logging-Dateien ausgeben.

2.7.3 Logging-Dateiformate

In der folgenden Tabelle finden Sie eine Übersicht über alle Ereignisse, die in der Logging-Datei aufgezeichnet werden. Bei aktiver Funktion *Interne Ereignisse loggen* im Konfigurationsdialog der Dateitrommel werden auch die vom Programm generierten internen Ereignisse (z.B. Busstatistikinformationen, Trigger, etc.) aufgezeichnet. Damit Busstatistikinformationen generiert werden, muß im Dialog *Karten- und Treiberoptionen* die entsprechende Option aktiviert sein. Dort stellen Sie auch ein, wie häufig diese Ereignisse generiert werden sollen.

In der ersten Spalte finden Sie alle Ereignistypen, die von Loggings aufgezeichnet werden. In der zweiten Spalte finden Sie das Format des jeweiligen Ereignisses im

ASCII-Logging. Die dritte Spalte gibt Auskunft, ob das Ereignis auch in binären MDF-Logging-Dateien erfaßt wird. Der letzten Spalte können Sie entnehmen, ob Sie die Funktion *Interne Ereignisse loggen* aktivieren müssen, um das Ereignis aufzuzeichnen.

Ereignistyp	Format im ASCII-Logging	binär	Bemerkungen
CAN-Botschaft	vgl. Trace-Fenster	ja	
Error-Frame	<Zeit> <CAN> `ErrorFrame`	ja	
Overload-Frame	<Zeit> <CAN> `Overloadframe`	ja	
CAN-Fehler	<Zeit> CAN <CAN> `Error:` <Fehlermeldung>	ja	Internes Ereignis
Busstatistik	<Zeit> <CAN> `Statistic:` <Daten>	ja	Internes Ereignis
Messungsstart	<Zeit> `Start der Messung`	nein	
Logging Trigger	`Statistic:` <Zeit> `log trigger event`	ja	Internes Ereignis
Triggerblock Anfang	`Begin Triggerblock` <Zeit und Datum>	nein	
Triggerblock Ende	`End Triggerblock` <Zeit und Datum>	nein	
Überlastsymbol	'*' am Anfang der Zeile nach der Datenverlust eingetreten ist.	nein	Internes Ereignis, vgl. Abschnitt 4.2

2.7.4 Datenanalyse im Offline-Modus

Um aufgezeichneten Logging-Dateien zu untersuchen, schalten Sie den CANalyzer mit dem Hauptmenüeintrag *Modus|Zu Offline* in den Offline-Modus. Abbildung 61 zeigt das zugehörige Datenflußdiagramm.

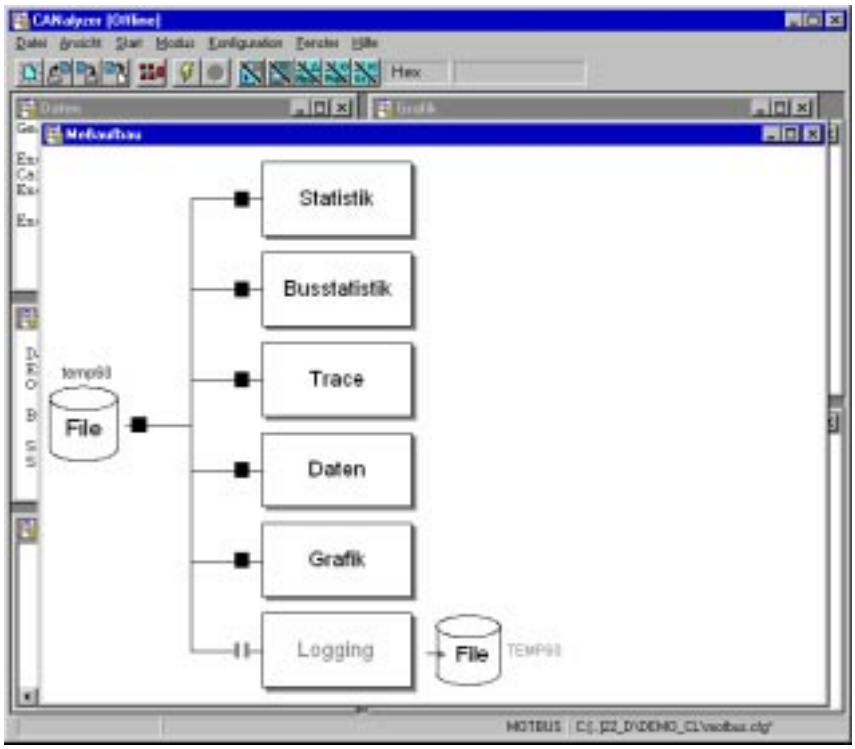


Abbildung 61: Datenflußdiagramm für den Offline-Modus

Als Datenquelle dient im Offline-Modus eine Datei, wie sie z.B. im Online-Modus durch das Logging erzeugt wurde. Alle Meß- und Auswertefenster stehen Ihnen im Offline-Modus analog zum Online-Modus zur Verfügung. Lediglich die Möglichkeit, Daten über den Bus zu versenden, entfällt. Zusätzliche bietet der Offline-Modus eine leistungsfähige Such- und Break-Funktion, mit der Sie die Wiedergabe der Logging-Datei gezielt anhalten können. Diese wird in Abschnitt 2.7 beschrieben. Im Logging-Block, der im Offline-Modus ebenfalls verfügbar ist, können die Daten in eine neue Datei umgespeichert werden, wobei durch die einfügbaren Datenmanipulationsblöcke eine gezielte Datenreduktion erreicht werden kann.

Mit dem Eintrag *Konfiguration...* im Kontextmenü der Datentrommel links im Offline-Meßaufbau geben Sie den Namen dieser Datei an. Der CANalyzer unterstützt dabei sowohl das binäre als auch das ASCII-Loggingformat.

Mit dem Eintrag *Break-Bedingungen...* darunter öffnen Sie einen Dialog, in dem Sie einen Unterbrechungspunkt – einen *Breakpoint* – setzten können. Die Offline-Wiedergabe wird bei Eintreten der von Ihnen angegebene Bedingung solange unterbrochen, bis Sie das Abspielen der Datei mit den Funktionen *Run* (F9), *Animate* (F8) oder *Step* (F7) fortsetzen.

Sie konfigurieren die Break-Bedingung mit der Schaltfläche *Bedingungen...*, die im Abschnitt 2.7.5 näher beschrieben werden. Sollten die dazu im Konfigurationsdialog vorgesehenen Mittel nicht ausreichen, bietet Ihnen die Programmiersprache CAPL mit der Funktion **`stop()`** die Möglichkeit, einen Breakpoint frei zu programmieren.

2.7.4.1 Ablaufsteuerung im Offline-Modus

Die folgenden Funktionen des Hauptmenüpunktes *Start* stehen Ihnen im Offline-Modus zur Verfügung, um das aufgezeichnete Busgeschehen am Bildschirm in Zeitlupe zu verfolgten:

Start

Die einzelnen Botschaften der Datenquelle werden ausgelesen und so schnell wie möglich durch die Komponenten des Meßaufbaus geschickt. Die Messung nach im Offline-Modus nach einer Unterbrechung fortgesetzt werden. Für einen Neustart muß *Reset* aufgerufen werden.

Reset

Nachdem eine Messung im Offline-Modus teilweise oder vollständig abgelaufen ist, kann sie mit Reset wieder auf den Anfang zurückgesetzt werden und somit von vorn untersucht werden.

Animate

Anstatt die Daten so schnell wie möglich aus der Quelldatei zu lesen, werden im Animate-Modus pro Sekunde nur ca. 3 Einträge aus der Quelldatei gelesen. Daraus resultiert eine Zeitlupendarstellung der Vorgänge. Dabei können alle Anzeigefenster aktiv sein, so daß z.B. im Trace-Fenster bequem die Botschaftsabfolge beobachtet werden kann. Der Animate-Lauf kann durch die Taste <Esc> oder den Menüpunkt *Start|Break* unterbrochen werden. Die Geschwindigkeit des Animate-Modus kann in der Datei `CAN.INI` im Abschnitt `[Offline]` die Zeile

`AnimationDelay=nnn`

ein, wobei *nnn* die Zeit zwischen dem Abspielen von zwei aufeinanderfolgenden Ereignissen im Millisekunden beschreibt. (Der Default-Wert ist 300 [ms])

Break

Der Menüpunkt unterbricht im Offline-Betrieb die Wiedergabe der Daten aus der Quelldatei (Animate-Lauf). Das gleiche kann durch die Taste <Esc> erreicht werden. Bei erneutem Start wird die Wiedergabe an der Stelle fortgesetzt, an der sie mit Break unterbrochen wurde.

Step

Mit diesem Menüpunkt (oder der Taste <F7>) läuft ein Einzelschritt der Messung ab. Es wird immer nur eine weitere Botschaft aus der Logging-Datei gelesen und im Datenflußplan verarbeitet.

Innerhalb einer Messung kann beliebig zwischen *Start*, *Animate* und *Step* hin- und hergeschaltet werden.

2.7.4.2 Konfiguration von Online- und Offline-Modus

Grundsätzlich gibt es für Online- und Offline-Modus unterschiedliche Meßaufbaubilder und völlig getrennte Parametersätze, so daß die beiden Modi - einschließlich des Fenster-Layouts - unterschiedlich konfiguriert werden können.

Beim Umschalten zwischen Online- und Offline-Modus möchte man jedoch oft die in einem Modus getroffenen Konfigurationseinstellungen in den anderen Modus übernehmen. Dazu stellt der CANalyzer die Funktionen *Online(Kopie)* und *Offline(Kopie)* im Hauptmenüpunkt *Modus* zur Verfügung.

Online

Wechselt in den Online-Modus. Im Online-Betrieb wird im Meßaufbaufenster ein Datenflußplan dargestellt, bei dem die CAN-Einsteckkarte als Datenquelle dient. Außerdem ist ein Sendeblock vorhanden. Die Zeitbasis im Online-Betrieb ist der reale Zeitverlauf, die Uhren im CANalyzer und im CAN-Board werden beim Start synchronisiert. Alle Zeitangaben beziehen sich auf den Start der Messung. Die Erfassung im Online-Modus kann nach einer Unterbrechung nicht weitergeführt sondern nur neu gestartet werden, der Animate-Modus ist nicht möglich.

Online (Kopie)

Wechselt aus dem Offline-Modus in den Online-Modus, wobei der Datenflußplan des Offline-Modus mit allen Funktionsblöcken übernommen wird. Der entsprechende Teil der letzten Online-Konfiguration geht dabei verloren. Da der Offline-Modus keinen Sendezweig hat, bleibt dieser Teil unverändert.

Offline

Wechselt in den Offline-Modus. Im Offline Betrieb wird im Meßaufbaufenster ein Datenflußplan dargestellt, bei dem eine Dateitrommel als Datenquelle dient. Durch *Start* wird die Auswertung dieser Datei gestartet, die Auswertung kann nach einer Unterbrechung mit <ESC> weitergeführt werden, der Animate- und Einzelschrittbetrieb ist möglich. Die Zeitbasis im Offline-Betrieb basiert auf den in der Datei festgehaltenen, zu den Daten gehörenden Zeiten.

Offline (Kopie)

Wechselt aus dem Online-Modus in den Offline-Modus, wobei der Datenflußplan des Online-Modus hinter der Abzweigung zum Sendeblock übernommen wird. Dies betrifft alle Fensterkonfigurationen und Funktionsblöcke. Die letzte Offline-Konfiguration geht dabei verloren.

Die Umschaltfunktionen mit Kopie sind ein bequemer Weg, um aufwendige Einstellungen wie Triggerbedingungen, Konfiguration der Datenanzeige oder CAPL-Programme zu übernehmen und sofort mit der Analyse bzw. mit einer neuen Aufzeichnung starten zu können.

2.7.5 Trigger- und Suchbedingungen

Für die Triggerung des Loggings, zum Suchen im Trace-Fenster und zum Unterbrechen einer Offline-Messung können Bedingungen formuliert werden. Die Eingabe dieser Bedingungen erfolgt stets in dem selben Dialog.

2.7.5.1 Bedingungsprimitive

Eine Bedingung setzt sich aus Bedingungsprimitiven zusammen, die mit dem logischen *oder* oder *und* verknüpft werden können.

Beispiel:

1.Primitive: ID > 100

2.Primitive: D0 == 0

Bei einer Verknüpfung mit *oder* werden alle Botschaften gefunden, deren ID größer als 100 oder deren erstes Datenbyte 0 ist. Bei der *und* Verknüpfung werden alle Botschaften gefunden, deren ID größer als 100 und deren erstes Datenbyte 0 ist.

Um ein Primitiv neu einzutragen stehen Ihnen die Buttons *Definieren* bzw. *Neues Prim.* (nur bei Verwendung einer symbolischen Datenbasis) zur Verfügung.

Die Dialoge zur Eingabe bzw. zur Änderung von Primitiven unterscheiden sich, abhängig davon ob Sie mit einer symbolischen Datenbasis (Button *Neues Prim.*) arbeiten oder nicht (Button *Definieren*). In der oberen Zeile können die gewünschten Attribute angegeben werden. Darunter werden der Identifier und die Datenbytes angegeben. Außerdem kann ein Zeitfenster spezifiziert werden, innerhalb dessen gesucht werden soll.

2.7.5.2 Eingabe von ID und Datenbytes

Sowohl für Identifier als auch für Datenbytes gelten die gleichen Regeln. In der Spalte nach der Objektbezeichnung ist ein Toggle-Feld (*Beziehung*), das den gewünschten Vergleich bestimmt. Ist es leer, wird das Objekt nicht beachtet. Soll ein Vergleich stattfinden, ist eines der folgenden Symbole auszuwählen:

Symbol	Bedeutung
==	gleich
!=	ungleich
<	kleiner
<=	kleiner oder gleich
>	größer
>=	größer oder gleich

Dahinter befindet sich eine auszuführende Maske, bei der zwischen dem verwendeten Zahlenformat unterschieden werden muß:

Zahlenformat dezimal

Hier wird der gewünschte Wert als Dezimalzahl. Um den Identifier 100 zu suchen, wird eingetragen:

ID == 100

Solange sich noch die Zeichen X (unbestimmt) oder $ (ungültig) im Eingabefeld befinden, ist die Eingabe ungültig.

Zahlenformat binär

Hier wird eine Maske angegeben, in der spezifiziert wird, welche Bit gesetzt sein sollen, welche nicht, und welche ausgeblendet, also nicht berücksichtigt werden sollen. Hierbei bedeuten die Zeichen:

X	ausblenden
1	Bit ist gesetzt
0	Bit ist nicht gesetzt

Um alle Botschaften zu finden, deren erstes Datenbyte ungerade ist, muß folgendes eingetragen werden:

D0 == XXXXXXX1

Hinweis: Die Verwendung von >, < etc. im Binärmodus ist möglich, aber bei maskierten Bedingungen schwer zu interpretieren und nur für Kenner der genauen maschineninternen Datenrepräsentation interessant.

Zahlenformat hexadezimal

Auch hier können Masken angegeben werden. Es können entweder ganze Zahlen in hexadezimaler Schreibweise eingetragen oder Nibbles (Halbbytes) ausgeblendet werden, indem X eingetragen wird. Das Zeichen $ steht für ungültig. Es kann z.B. durch folgenden Fall entstehen:

Im Binärmodus ist XXXXXXX1 eingetragen, dann wird in den Hex-Modus gewechselt. Die linken vier X werden als Nibble in ein X übersetzt. Die unteren Werte XXX1 können nicht als vollständig bestimmtes Nibble interpretiert werden. Das Ergebnis ist also X$.

2.7.5.3 Symbolische Triggerung

Bei Verwendung einer symbolischen Datenbasis kann auf Botschaften und auf Signale getriggert werden.

Über die Schaltflächen *SYM* hinter den Texteingabefeldern *Botschaftsname* und *Signalbezeichnung* wird der symbolische Auswahldialog aufgerufen, in dem Sie die Botschaft und ggf. auch das Signal aus der Datenbasis auswählen können, auf das getriggert werden soll.

Geben Sie im symbolischen Auswahldialog nur einen Botschaftsnamen ein, so wird die Aufzeichnung getriggert, immer wenn die Botschaft am Bus auftritt.

Wenn Sie im darunterliegenden Eingabefeld einen Signalnamen auswählen, wird anhand des Signalwertes entschieden, wann die Aufzeichnung getriggert wird. Der Vergleich der Signalwerte erfolgt anhand ihrer physikalischen Interpretation. Hierzu ist ein Vergleichsoperator auszuwählen und ein Zahlenwert einzutragen.

Die Speicherung der physikalischen Größen erfolgt in den CAN-Botschaften in diskreter Form. Daher ist der angegebene Zahlenwert nicht immer auf einen diskreten Wert abbildbar. In diesem Fall werden nach Drücken der Eingabetaste oder der Schaltfläche *OK* die beiden nächstmöglichen physikalischen Werte in zwei Feldern unter Ihrer Eingabe dargestellt.

Beispiel:

Signal Motordaten.Drehzahl sei definiert als 16 Bit unsigned mit einem Offset von 0 und einem Faktor von 10. Wird als zu vergleichender Wert 1015 angegeben, müßte der Rohwert 1015/10 = 101.5 sein. Da nur diskrete Werte vorkommen, muß entweder 101 oder 102 verwendet werden, was physikalischen Größen von 1010 oder 1020 entspricht. Genau diese beiden Zahlen erscheinen in den beiden Info-Feldern.

Trotz diskreter Speicherung können gültige Größen Nachkommastellen haben. Wird im obigen Beispiel mit einem Faktor von 10.5 gerechnet, so werden als mögliche Werte 1008 und 1018.5 vorgeschlagen.

Für jedes Primitiv können Sie zusätzlich einen Zeitrahmen angeben, innerhalb dessen es aktiv sein soll. Außerhalb dieses Zeitfensters ist die Bedingung dann niemals erfüllt.

3 CAPL - Programmierung

3.1 Übersicht

Die universelle Einsetzbarkeit des CANalyzers bzw. von CANoe ergibt sich u.a. aus der Möglichkeit der freien Programmierung.

Die CAN Access Programming Language CAPL ist eine C-ähnliche Programmiersprache, die es Ihnen ermöglicht, den CANalyzer und CANoe für individuelle Anwendungen zu programmieren. So entsteht z.B. bei der Entwicklung von Netzknoten das Problem, daß für Tests die restlichen Busteilnehmer noch nicht verfügbar sind. Zur Emulation der Systemumgebung kann mit Hilfe von CAPL z.B. der Datenverkehr aller restlichen Stationen nachgebildet werden.

Sie können mit CAPL auch Programme zur problemangepaßten Analyse des Datenverkehrs schreiben oder ein Gateway - ein Verbindungselement zwischen zwei Bussen - programmieren, um Daten zwischen verschiedenen CAN-Bussen auszutauschen.

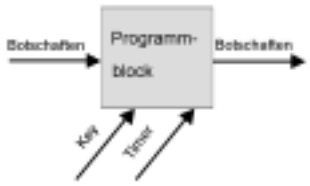
CAPL-Knoten fügen Sie im Datenflußplan als Funktionsblöcke ein. Als Eingang in CAPL dienen Ereignisprozeduren, mit denen Sie auf äußere Ereignisse (wie z.B. auf das Auftreten bestimmter Botschaften) reagieren können. Sie senden Botschaften, indem Sie die Funktion **`output()`** aufrufen. Diese Sprachmittel und der symbolische Zugriff auf die verschiedenen in der Datenbasis definierten Variablen ermöglichen die einfache prototypische Erstellung von Modellen der Knoten. Die Ereignisprozeduren können in einem komfortablen Browser bearbeitet werden.

In Abschnitt 3.1.1 werden zunächst die Einsatzmöglichkeiten von CAPL-Programmen betrachtet. Wenn Sie zum ersten Mal mit CAPL arbeiten, bietet Ihnen der folgende Abschnitt 3.2 eine Einführung in die Programmiersprache. Ein kleines aber vollständiges Beispiel für das Erstellen eines CAPL-Programms finden Sie in Abschnitt 3.3. Abschnitt 3.4 gibt danach eine Übersicht über den CAPL-Browser, das im Standardlieferumfang enthaltene Werkzeug zum Erstellen und Ändern von CAPL-Programmen.

Im Abschnitt 3.5 folgt schließlich eine Sprachbeschreibung als Referenz. Alle Verweise auf Umgebungsvariable oder CAPL-Funktionen für Umgebungsvariable gelten *nicht* für den CANalyzer. Umgebungsvariable werden ausschließlich von CANoe unterstützt.

3.1.1 Einsatzmöglichkeiten von CAPL-Programmen

CAPL-Programme haben einen Eingang, über den Botschaften als Events in den Programmblock gehen. Am Ausgang erscheinen alle Botschaften die durch das Programm weitergegeben oder erzeugt werden. Der Programmblock kann zusätzlich auf Tastatureingaben (Key), Zeitereignisse (Timer) und - bei CANoe - auf Änderungen von Umgebungsvariablen wie Schalter oder Schiebereglerpositionen reagieren.



Sie können ein CAPL-Programm also einsetzen, um für Ihre spezielle Aufgabenstellung Prüfungen und Tests zu entwickeln. Das CAPL-Programm reagiert dabei auf Botschaften, die der CANalyzer bzw. CANoe am CAN-Bus registriert, und kann daraufhin eigene Analyse- und Testfunktionen aufrufen.

Sie können ein CAPL-Programm auch einsetzen, um die Systemumgebung eines Steuergerätes zu emulieren. Das CAPL-Programm reagiert dabei sowohl auf Botschaften am CAN-Bus als auch auf Ihre Tastatureingaben, um je nach registriertem Ereignis mit bestimmten CAN-Botschaften zu antworten. Welche Aktionen auf welche Ereignisse ausgeführt werden, liegt wiederum vollständig in Ihrer Hand.

Eine weitere Einsatzmöglichkeit von CAPL besteht darin, ein Gateway - also ein Verbindungselement zwischen zwei Bussen - zu programmieren, um Daten zwischen verschiedenen CAN-Bussen auszutauschen und darüber hinaus fehlerhafte Daten bei diesem Austausch zu korrigieren.

Nicht zuletzt kann auch die Triggerung des Logging-Blocks über ein CAPL-Programm erfolgen. Durch die CAPL-Programmierung können beliebig komplexe Bedingungen für die Triggerung formuliert werden. Die Triggerung wird dabei durch den Aufruf der Intrinsic-Funktion **`trigger()`** ausgelöst.

3.1.2 Einbinden von CAPL-Programmen

Ein CAPL-Programm kann im Meßaufbau an allen Hot-Spots und zusätzlich im Simulationsaufbau von CANoe direkt am Bussymbol eingefügt werden. Wählen Sie dazu im Kontextmenü des Hot-Spots den Menüpunkt Füge *CAPL-Knoten ein* und geben Sie im Konfigurationsdialog den Namen der CAPL-Programmdatei an, die Sie diesem Knoten zuordnen wollen. Wenn Sie ein neues CAPL-Programm erstellen wollen, können Sie hier auch den Namen einer noch nicht vorhandenen Datei eintragen. Beim Editieren wird diese Datei dann automatisch angelegt.

Mit der Schaltfläche *Edit...* im Konfigurationsdialog oder mit einem Doppelklick auf einen CAPL-Knoten öffnen Sie den CAPL-Browser, mit dem Sie CAPL-Programme erstellen und modifizieren können.

Hinweis: Falls Sie Ihren eigenen Editor zum Bearbeiten von CAPL-Programmen vorziehen, tragen Sie diesen in der Datei `CAN.INI` im Abschnitt `[Environment]` ein.

Bevor Sie die Messung starten, müssen Sie alle CAPL-Programme der Konfiguration compilieren. Sie können den CAPL-Compiler aus dem CAPL-Browser oder aus dem

Konfigurationsdialog heraus starten. Um alle Knoten auf einmal zu compilieren, wählen Sie einfach dem Hauptmenüeintrag *Konfiguration|Alle Knoten compilieren*.

Beachten Sie, daß ein CAPL-Programme durchaus unterschiedlich reagiert, je nachdem an welcher Stelle im Meßaufbau Sie es einfügen. So lassen sich in einem CAPL-Programm rechts vom Sendezweig zwar Botschaften generieren, aber nicht auf den Bus senden. Da der Datenfluß von links nach rechts gerichtet ist, werden diese Botschaften nur in die Funktionsblöcke rechts vom CAPL-Programm weitergeleitet. Auf den Bus können Botschaften nur von CAPL-Programmen aus gesendet werden, die sich im Sendezweig[4] des CANalyzers bzw. im Simulationsaufbau von CANoe befinden. Dieses zunächst überraschende, aber durchaus logische Verhalten gilt genauso für den Generatorblock, der im rechten Teil des Meßaufbaus ebenfalls Botschaften generiert, ohne den Bus zu beeinflussen.

Im allgemeinen bietet es sich daher an, CAPL-Programmblöcke, die ausschließlich der Analyse dienen, auf der rechten Seite des Meßaufbaus einzufügen, während Programmblöcke zum Senden von CAN-Botschaften im Sendezweig des CANalyzers bzw. im Simulationsaufbau von CANoe eingefügt werden müssen.

3.1.3 Einsatz der symbolischen Datenbasis in CAPL

Wie von den anderen Funktionsblöcken im Meßaufbau, haben Sie auch aus CAPL Zugriff auf die symbolischen Informationen aus der Datenbasis. Beispielsweise können Sie, statt den Identifier 100 zu verwenden, in Ihrem CAPL-Programm an allen Stellen den symbolischen Namen *Motordaten* benutzen, wenn Sie in Ihrer Datenbasis dem Identifier 100 diesen Namen zugeordnet haben.

Die Verwendung der symbolischen Datenbasis macht Ihre Programme weitgehend unabhängig von Informationen, die nur für das CAN-Protokoll von Belang sind, für die Applikationen dagegen keine Bedeutung haben. Nehmen Sie beispielsweise an, daß Sie während der Entwicklungsphase feststellen, daß bestimmte CAN-Identifier Ihres Systems aus Priorisierungsgründen neu vergeben werden müssen, und etwa die Botschaft *Motordaten* Ihres Systems statt des Identifiers 100 nun den höherprioren Identifier 10 erhält.

Falls Sie in diesem Fall bereits Testkonfigurationen und CAPL-Programme für Ihr System entwickelt haben, die ausschließlich auf der symbolischen Information aufbauen (die also nirgends den Identifier 100 verwenden, sondern stets den Namen *Motordaten*), können Sie nach dem Anpassen der Datenbasis die neuen Informationen in die Konfiguration übernehmen, indem Sie die CAPL-Programme erneut compilieren. Das Anpassen der CAPL-Programme an die neuen Identifier entfällt, da Sie in den Programmen ja nur die symbolischen Namen (*Motordaten*), nicht aber die CAN-Identifier (vorher Id 100, jetzt Id 10) verwendet haben.

Es ist daher empfehlenswert, alle Informationen, die lediglich den CAN-Bus betreffen, in der Datenbasis zu verwalten, und im CANalyzer bzw. in CANoe ausschließlich die applikationsrelevanten symbolischen Informationen zu verwenden.

[4] Auch CAPL-Programme, die an Hot-Spots links von Sendezweig des CANalyzers eingefügt werden, können Botschaften auf den Bus senden. Der Teil des Datenflusses zwischen Kartensymbol und Abzweigung zum Sendeblock muß strenggenommen also zum Sendezweig dazugerechnet werden.

3.2 Einführung in CAPL

CAPL ist eine prozedurale Sprache, bei der die Ausführung von Programmblöcken durch Ereignisse gesteuert wird. Diese Programmblöcke werden als Ereignisprozeduren bezeichnet.

Der Programmcode, den Sie in Ereignisprozeduren definieren, wird bei Eintreten des Ereignisses ausgeführt. Sie können so z.B. auf Tastendruck (**`on key`**) eine Botschaft auf den Bus legen, das Auftreten von Botschaften am Bus mitverfolgen (**`on message`**), oder aber zyklisch bestimmte Aktionen (**`on timer`**) ausführen.

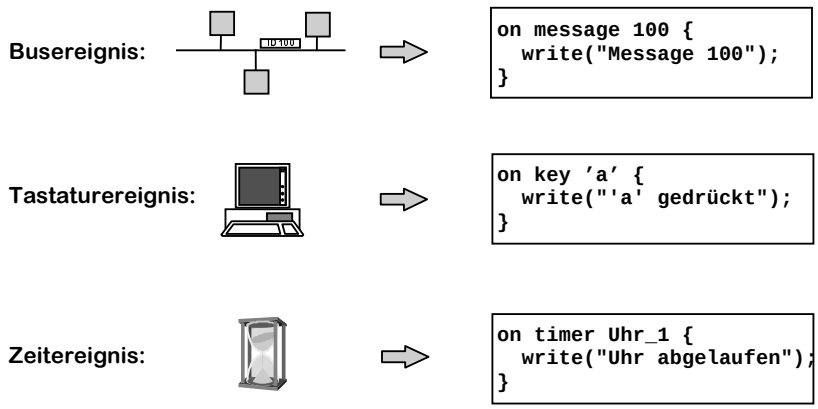


Abbildung 62: Beispiele für CAPL-Ereignisprozeduren[5]

Ein CAPL-Programm besteht aus zwei Teilen:

1. Deklaration und Definition globaler Variablen
2. Deklaration und Definition von benutzerdefinierten Funktionen und Ereignisprozeduren

Variablen in CAPL und das Ereignisprozedurkonzept werden in den folgenden Abschnitten näher beschrieben.

3.2.1 CAPL Variablen

Als Datentypen für Variablen stehen Ganzzahlen (**`dword`**, **`long`**, **`word`**, **`int`**, **`byte`**, **`char`**), Gleitkommazahlen (**`float`** und **`double`**), CAN-Botschaften (**`message`**) und Timer (**`timer`** bzw. **`msTimer`**) zur Verfügung. Bis auf die Timer können alle Variablen bereits bei der Deklaration initialisiert werden.

[5] Neben Tastaturereignissen können Sie in CANoe mit Ereignisprozeduren des Typs **`on envvar`** auch noch auf Aktionen reagieren, die Sie an selbstdefinierten Bedienpanels ausführen.

```
variables {
   int msgCount;     // wird bei Messungsstart auf 0 gesetzt
   message 34 sendMsg = { // Botschaft mit Id 34 deklarieren
      dlc = 1;            // Data Length Code = 1 setzen
      byte(0) = 1;        // 1. Datenbyte = 1 setzen
   }
}
```

Ganzzahlen und Gleitkommazahlen werden wie in anderen Programmiersprachen verwendet. Sie sind untereinander verträglich, die Arithmetik erfolgt mit 32 Bit Auflösung bei Ganzzahlen und 80 Bit bei Gleitkommazahlen.

Im Browser wird die Deklaration globaler Variablen in der Globalvariablen-Pane rechts oben vorgenommen. Die Datentypen **dword**, **long**, **word**, **int**, **byte** und **char** können analog zu der Programmiersprache C verwendet werden. Die Datentypen **float** und **double** sind Synonyme und bezeichnen 64 Bit Gleitkommazahlen nach dem IEEE Standard.

Variablen können bei der Deklaration initialisiert werden. Hierfür ist sowohl die einfache Schreibweise als auch die Klammerung mit **{}** möglich. Der Compiler initialisiert alle Variablen mit Ausnahme von Timern mit Defaultwerten (i.A. **0**).

Lokale Variablen werden in CAPL im Gegensatz zu C immer statisch angelegt. Das bedeutet, daß eine Initialisierung nur beim Programmstart durchgeführt wird. Die Variablen besitzen beim nächsten Eintritt in die Prozedur den Wert, den sie beim letzten Verlassen der Prozedur hatten.

Die Deklaration von Feldern (Arrays, Vektoren, Matrizen) ist in CAPL analog zur Programmiersprache C erlaubt.

```
variables {
   int vector[5] = {1,2,3,4,5};
   int matrix[2][2] = {{11,12},{21,22}};
   char progname[10] = "CANalyzer";
}
```

Felder werden formal wie in C übergeben. Als Erweiterung ist aber auch die Übergabe mehrdimensionaler Arrays möglich. Die Anzahl der Elemente einer Dimension läßt sich mit der Funktion **elCount()** erfragen. Der Compiler führt eine Wertebereichsüberprüfung durch. Wird ein Index kleiner Null oder größer als die Zahl der Elemente verwendet, wird über die Funktion **runError()** ein Laufzeitfehler erzeugt.

Variablen vom Typ **timer** (auf Sekundenbasis) bzw. **msTimer** (auf Millisekundenbasis) dienen zur Erzeugung von Zeitereignissen. Timer-Variablen werden bei Programmstart nicht automatisch initialisiert, sondern müssen im Programm explizit mit der Funktion **setTimer()** „aufgezogen“ werden.

```
variables {
   timer delayTimer; // Deklaration eines Sekunden-Timers ...
   msTimer cycTimer; // ... und eines Millisekunden-Timers
}

...
setTimer(delayTimer,3); // Timer auf 3 sec aufziehen
setTimer(cycTimer,100); // Timer auf 100 msec aufziehen
...
```

CAPL erlaubt im Gegensatz zu C keine Verwendung von Pointern. Auf dieses Sprachmittel wurde bewußt verzichtet, um CAPL-Programme sicherer zu machen. Speicherschutzverletzungen können so z.B. wesentlich besser abgefangen werden: Statt eines Absturzes in der hochprioren Echtzeitbibliothek des Programmes, wird die Messung bei Programmierfehlern kontrolliert beendet und eine Meldung im Write-Fenster ausgegeben.

Da der Schwerpunkt bei der Modellierung mit CAPL in der Beschreibung des Busverhaltens der Knoten liegt, ist der Sprachumfang von CAPL normalerweise ausreichend. Dennoch lassen sich auch umfangreichere Funktionen in C, C++ oder PASCAL in einer eigenen Windows DLL implementieren und in CAPL einbinden. Nähere Hinweise dazu finden Sie in Abschnitt 3.6.

3.2.2 Deklaration von Botschaften

Mit dem Schlüsselwort **message** werden Botschaften deklariert, die aus dem CAPL-Programm ausgegeben werden sollen. Zur vollständigen Deklaration gehört der Botschafts-Identifier bzw. beim Arbeiten mit symbolischen Datenbasen der Botschaftsname. Schreiben Sie also z.B.

```
message 0xA m1;          // Botschaftsdeklaration (hex)
message 100 m2;          // Botschaftsdeklaration (dec)
message Motordaten m3;   // symbolische Deklaration
message * wcrd;          // Deklaration ohne Id
...
output(m1);              // Sende Botschaft m1
output(m2);              // Sende Botschaft m2
output(m3);              // Sende Botschaft m3
wcrd.id = 0x1A0;         // Id festlegen ...
output(wcrd);            // ... und Botschaft senden
```

um die Botschaften mit den Identifier A (hex) und 100 (dec) sowie die in der Datenbasis definierte Botschaft *EngineData* auf den Bus auszugeben.

Als Bezeichnung von Botschaften wird also die Botschaftskennung (Botschafts-Identifier) als Zahl (integer) in dezimaler oder hexadezimaler Darstellung angegeben. Für Extended Identifier wird an den Identifier ein **x** angehängt. Die Angabe von ***** bedeutet, daß diese Variable zunächst noch keine Botschaftskennung enthält. Diese muß dann in anderer Weise vor dem Versenden des Objektes festgelegt werden. Solche Objekte dienen z.B. bei Filteraufgaben zur Speicherung aller Objekte, die

unverändert weitergereicht werden sollen. (Die empfangene Botschaft wird dann einschließlich des Botschafts-Identifiers kopiert.)

Der Zugriff auf den Datenbereich einer Botschaft erfolgt über die Angabe das Datentyps als Datenselektor und den byte-wertigen Offset (bei 0 beginnend). Als Datenselektoren können **long**, **dword**, **int**, **word**, **char** und **byte** verwendet werden.

Beispielsweise senden Sie mit

```
message 100 msg;
msg.DLC = 1;
msg.BYTE(0) = 0xff
output(msg);
```

eine Botschaft 100 mit DLC = 1 und dem ersten Datenbyte 0xFF auf den Bus.

Hinweis: CAPL-Programme, mit denen Sie Botschaften auf den CAN-Bus senden wollen, *müssen* im CANalyzer immer im Sendezweig des Meßaufbaus bzw. in CANoe im Simulationsaufbaufenster einfügt werden. Botschaften, die von CAPL-Programmen weiter rechts im Meßaufbau gesendet werden, werden *nicht* auf den Bus ausgegeben.

Der Zugriff auf die Komponenten der Objekte erfolgt über Komponenten-Selektoren. Wollen Sie ein Objekt (bei CAN-Karten mit mehr als einem CAN-Chip) für einen bestimmten Chip definieren, so stellen Sie den entsprechenden Selektor (CAN1 bzw. CAN2) gefolgt von einem Punkt (.) der Botschaft voran.

Mit den folgenden Komponenten-Selektoren greifen Sie auf die Kontrollinformationen der CAN-Botschaftsobjekte zu:

ID	Botschafts-Identifier
CAN	Bausteinnummer
DLC	Data-Length-Code
DIR	Übertragungsrichtung, mögliche Werte: **RX**, **TX**, **TXREQUEST**.
RTR	Remote-Transmission-Request; mögliche Werte: **0** (kein RTR), **1** (RTR)
TYPE	Kombination aus **DIR** und **RTR** für effiziente Auswertung. (**TYPE = (RTR << 8) | DIR**)
TIME	Zeitpunkt, Einheit: 10 Mikrosekunden

Beispiele für den Einsatz von Komponenten-Selektoren für Botschaften finden Sie in der Online-Hilfe des CAPL-Browsers.

Hinweis für CANoe-Anwender:

Mit Hilfe des **DIR**-Selektors können Sie im Meßaufbau unterscheiden, ob eine Botschaft von einem externen Netzknoten (**DIR == RX**) oder aber aus einem CAPL-Programm im Simulationsaufbau (**DIR == TX**) gesendet worden ist.

Da die Netzknoten in CANoe jedoch als *unabhängige* Einheiten simuliert werden, ist diese Semantik im Simulationsaufbau von CANoe geändert: CAPL-Knoten im Simulationsaufbau empfangen (bis auf den Sender) alle Botschaften mit **DIR == RX**, unabhängig davon, ob der Sender real ist oder von CANoe simuliert wird. Zur Unterscheidung, ob die empfangene Botschaft von einer realen oder von einer simulierten Komponente gesendet wurde, steht Ihnen der Selektor **SIMULATED** zur Verfügung.

```
on message LightState {
   if (this.dir == RX) {
      if (!this.SIMULATED) {
         write("LightState von realem System empfangen");
      }
      putValue(Bulb,this.OnOff);
   }
}
```

3.2.3 Ereignisprozeduren

Auf folgende Ereignisse können Sie in CAPL mit Ereignisprozeduren reagieren:

Ereignis	Ereignisprozedur
Empfang einer CAN-Botschaft	**on message{}**
Drücken einer Taste	**on key{}**
Initialisieren der Messung (vor Messungsstart)	**on preStart{}**
Messungsstart	**on start{}**
Messungsende	**on stopMeasurement{}**
CAN-Controller geht in ErrorActive	**on errorActive{}**
CAN-Controller geht in ErrorPassive	**on errorPassive{}**
CAN-Controller erreicht das Warning Limit	**on warningLimit{}**
CAN-Controller geht Bus Off	**on busOff{}**
Ablaufen eines Timers	**on timer{}**
Auftreten eines Error-Frames	**on errorFrame{}**
Umgebungsvariablenänderung (nur CANoe)	**on envVar{}**

Tritt das spezifizierte Ereignis ein, wird der zugehörige Prozedurkörper ausgeführt. Dieser kann die Deklaration von lokalen Variablen, C-ähnliche Arithmetik- und Kontrollanweisungen und den Aufruf von eingebauten Standardprozeduren enthalten.

Auf Botschaften reagieren

Um in den CAPL-Knoten auf das Empfangen von CAN-Botschaften zu reagieren, steht Ihnen in der Ereignisprozedurtyp **on message** zur Verfügung.

on message 123	Reagiere auf Botschaft 123 (dec), Empfangschip wird nicht berücksichtigt
on message 0x123	Reagiere auf Botschaft 123 (hex), Empfangschip wird nicht berücksichtigt
on message EngineData	Reagiere auf Botschaft `EngineData`
on message CAN1.123	Reagiere auf Botschaft 123, wenn sie von Chip CAN1 empfangen wird
on message *	Reagiere auf alle Botschaften
on message CAN2.*	Reagiere auf alle Botschaften die von Chip CAN2 empfangen werden
on message 100-200	Reagiere auf alle Botschaften mit Identifiern zwischen 100 und 2

Innerhalb einer **on message**-Prozedur steht Ihnen das Schlüsselwort **this** zur Verfügung, um auf die Daten der gerade empfangenen Botschaft zuzugreifen.

Im folgenden Beispiel wird mit CAPL ein einfaches Gateway programmiert. Das Gateway soll alle Botschaften zwischen Bus 1 und Bus 2 in beiden Richtungen übertragen, die Botschaft mit Id 34 jedoch verfälschen, indem Byte 4 auf 0 gesetzt wird.

```
on message CAN1.34 {
   message CAN2.34 sendMsg; // lokale Botschaftsvariable mit
                            // Namen sendMsg, Identifier 34,
                            // Zielcontroller CAN 2
   sendMsg = this;          // Alle Daten und Attribute von
                            // empfangener Botschaft (this)
                            // in die zu sendende kopieren
   sendMsg.byte(4) = 0;     // Byte 4 verfälschen,
                            // stets 0 eintragen
   output(sendMsg);         // Botschaft senden
}
```

Beachten Sie, daß es an dieser Stelle nicht reichen würde, das vierte Byte direkt an der empfangenen Boschaft **this** zu verändern. Wie in Abschnitt 3.2.5 beschrieben, sind alle Änderungen von **this** lokal und werden daher in der Funktion **output()** nicht berücksichtigt. **this** muß folglich hier in eine lokale Botschaftsvariable umkopiert werden.

Für alle weiteren Botschaften wird eine Bridge-Funktion implementiert: Alle Botschaften von Bus 1 werden zu Bus 2 übertragen und umgekehrt.

```
on message CAN1.* { // mit '*' werden alle Botschaften
                    // bezeichnet, die in keiner anderen
                    // Prozedur verarbeitet werden.
   message CAN2.* sendMsg;
   sendMsg = this;
   output(sendMsg);
}

on message CAN2.* {
   message CAN1.* sendMsg;
   sendMsg = this;
   output(sendMsg);
}
```

CAPL-Programme sind defaultmäßig für Busereignisse nicht durchlässig. Dies bedeutet, daß Sie mit einem CAPL-Knoten im Auswertezweig des Meßaufbaus den Datenfluß auf der rechten Seite des Knotens sperren. Sie müssen das Weiterreichen von Botschaften in CAPL-Knoten im Auswertezweig explizit programmieren. Schreiben Sie

```
on message * {
    output(this);
};
```

um den CAPL-Knoten für Botschaften durchlässig zu machen. Beachten Sie jedoch, daß Sie eine Rekursion programmieren, wenn Sie diesen Code im Sendezweig des CANalyzers bzw. im Simulationsaufbau von CANoe verwenden: Auf jede empfangene Botschaft wird hier reagiert, indem die gleiche Botschaft sofort erneut auf den Bus gesendet wird, worauf wieder die Ereignisprozedur **`on message *`** aufgerufen wird, usw..

Hinweis: (für CANoe-Anwender) Im Unterschied zu Busereignissen, sind CAPL-Programme im Auswertezweig für Werteänderungen von Umgebungsvariablen durchlässig. Ein neuer Umgebungsvariablenwert steht daher auch auf der rechten Seite von CAPL-Knoten zur Verfügung, ohne das Sie dies explizit programmieren.

Für jede empfangene Botschaft wird in CAPL maximal eine Prozedur aufgerufen. Existieren in einem CAPL-Programm mehrere Prozeduren, die zur Botschaft passen, wird diejenige beauftragt, die den spezifischsten Namen hat. Dabei gelten folgende Regeln:

- Angabe einer Controller-Nummer hat Vorrang vor einer fehlenden Angabe
- Angabe einer Botschaftsnummer hat Vorrang vor der Angabe eines Jokers

Beispiel für Prioritätsreihenfolge (für eine Botschaft wird von oben nach unten die erste anwendbare Prozedur gesucht):

```
on message CAN1.123 // reagiert nur auf Botschaft 123
                    // von CAN1
on message CAN2.*  // reagiert auf alle Botschaften von CAN2
on message 123     // wird nie aufgerufen
on message*        // reagiert auf alle Botschaften von CAN1
                   // außer auf Botschaft 123
```

Die Ordnung im Beispiel entspricht der Darstellung der Event-Prozeduren im Browser. Zwei Event-Prozeduren mit dem gleichen Bezeichner sind in CAPL nicht erlaubt.

Auf Tastaturereignisse reagieren

Mit **on key**-Prozeduren können Sie auf Tastendruck bestimmte Aktionen ausführen.

on key 'a'	Reagiere auf Drücken der 'a'-Taste
on key ' '	Reagiere auf Drücken der Leertaste
on key 0x20	Reagiere auf Drücken der Leertaste
on key F1	Reagiere auf Drücken der F1-Taste
on key ctrlF12	Reagiere auf Drücken von Strg-F12
on key PageUp	Reagiere auf Drücken von Seite nach oben
on key Home	Reagiere auf Drücken von Pos 1
on key *	Reagiere auf jeden Tastendruck

Der Code für einen Tastendruck kann entweder als Zeichen, als Zahl oder mit Hilfe vordefinierter Bezeichnungen für Funktionstasten eingegeben werden.

Im folgenden Beispiel werden alle Botschaften mit dem (hexadezimalen) Identifier 1A1 gezählt und diese Zahl auf Drücken der Taste 'a' im Write-Fenster ausgegeben. Beachten Sie, daß dieses Programm für den Auswertezweig des Meßaufbaus geschrieben und dort mit dem Aufruf von **output(this)** für alle Botschaften durchlässig gemacht wurde. Ohne **output(this)** würden die Botschaften nicht an die weiter rechts stehenden Funktionsblöcke weitergereicht und dementsprechend natürlich auch in den zugeordneten Fenstern nicht dargestellt.

```
variables {
   int counter = 0;
}

on key 'a' {
   write("Es wurden %d Botschaften 0x1A1 gezaehlt",counter);
}

on message 0x1A1 {
   counter++;
   output(this); // nur im Auswertezweig
}

on message * {
    output(this); // nur im Auswertezweig !!!
}
```

Wenn Sie dieses CAPL-Programm unverändert in den Sendezweig des CANalyzers bzw. den Simulationsaufbau von CANoe einfügen, erhalten Sie eine Rekursion. (Auf jede empfangene Botschaft wird reagiert, indem die gleiche Botschaft sofort erneut auf den Bus gesendet wird, worauf wieder die Ereignisprozedur **on message *** aufgerufen wird, usw..)

Hier müssen Sie an beiden Stellen den Aufruf von **output(this)** auskommentieren, damit das Programm wie gewünscht funktioniert.

In **on key**-Prozeduren dient das Schlüsselwort **this** zur Ermittlung des aktuellen Tastencodes. Sie können also z.B. folgenden Code schreiben, um die Tastaturbehandlung in einer Ereignisprozedur zusammenzufassen.

```
on key * {
    switch(this) {
       case 'a' : ... break;
       case F10: ... break;
       ...
    }
}
```

Auf Werteänderungen von Umgebungsvariablen reagieren

Um in den Netzknotenmodellen von CANoe auf Werteänderungen von Umgebungsvariablen zu reagieren, steht Ihnen in der Ereignisprozedurtyp **on envVar** zur Verfügung. Im Gegensatz zu Botschaften werden Umgebungsvariablen in einem Datenflußzweig des Meßaufbaus durch einen CAPL-Knoten nicht gesperrt. Bei zwei hintereinandergeschalteten CAPL-Knoten reagieren daher beide mit der Ereignisprozedur **on envVar** auf dieselbe Umgebungsvariable. Mit den Funktionen **getValue()** und **putValue()** können Sie im Simulationsaufbau die aktuellen Umgebungsvariablenwerte abfragen bzw. setzen. Beachten Sie jedoch, daß Sie im Meßaufbaufenster die Funktion **putValue()** nicht aufrufen können.

In der Prozedur **on envVar X { ... }** wird auf das Ereignis Änderung der Umgebungsvariable **X** reagiert. Mit dem Schlüsselwort **this** kann innerhalb der

Ereignisprozedur auf die Umgebungsvariable zugegriffen werden, um z.B. mit der Funktion **getValue()** den Wert der Variablen zu bestimmen.

Im folgenden Beispiel wird auf die Änderung der Umgebungsvariablen *Schalter* reagiert und der Wert dem Signal *Stop* der Botschaft *Steuerung* zugewiesen:

```
on envvar Schalter {
   // Deklaration einer zu versendenden CAN-Botschaft
   message Steuerung msg;

   // Auslesen des Wertes des Schalters
   // Zuweisung an das Signal Stop
   msg.Stop = getValue(this);
   // Ausgabe der Botschaft auf den Bus
   output(msg);
}
```

Auf Zeitereignisse reagieren

In CAPL können Sie Zeitereignisse definieren. Beim Eintritt dieses Ereignisses, d.h. nach Ablauf einer bestimmten Zeitspanne wird dann die zugehörige **on timer**-Prozedur aufgerufen. Sie programmieren zyklische Programmabläufe, indem Sie innerhalb dieser **on timer**-Prozeduren das gleiche Zeitereignis erneut setzten.

Ziehen Sie mit der Funktion **setTimer()** einen Timer auf, den Sie vorher definiert haben. In CAPL stehen Ihnen dazu Timer auf Sekundenbasis (**timer**) und auf Millisekundenbasis (**msTimer**) zur Verfügung. Nach Ablaufen des Timers wird die zugehörige **on timer**-Prozedur aufgerufen. Die Maximalzeit beträgt 1799 sec für Variablen vom Typ **timer** und 65,535 msec für Variablen vom Typ **msTimer**. Mit der Funktion **cancelTimer()** können Sie einen bereits aufgesetzten Timer stoppen und so verhindern, daß die zugehörige **on timer**-Prozedur aufgerufen wird.

Im folgenden Beispiel wird die Botschaft 100 immer 20 Millisekunden nach Drücken der Taste 'a' auf den Bus gelegt.

```
msTimer myTimer;    // Millisekunden-Timer definieren
message 100 msg;    // zu sendende Botschaft definieren
...
on key 'a' {        // reagiere auf Tastendruck 'a' ...
   setTimer(myTimer,20); // ... ziehe Timer auf 20 ms auf
}
...
on timer myTimer {  // nach Ablaufen des Timers ...
   output(msg);     // ... sende Botschaft
}
```

Auf Systemereignisse reagieren

Die Ereignisprozeduren **on preStart**, **on start** und **on stopMeasurement** werden vor dem Start, beim Start bzw. beim Beenden der Messung aufgerufen. Hier

können Sie z.B. Variablen initialisieren, mit den Funktionen **fileRead*XXX*()** bzw. **seqFile*XXX*()** Initialisierungsdaten aus Dateien einlesen, mit der Funktion **write()** eine Startmeldung ins Write-Fenster ausgeben, Timer aufziehen oder nach Messungsende eine Statistik ausgeben.

Weitere Hinweise und Beispiele für den Einsatz der Systemereignisprozeduren finden Sie in der Online-Hilfe des CAPL-Browsers.

Auf Ereignisse des CAN-Controllers reagieren

Die Ereignisprozeduren **on errorActive**, **on errrorPassive**, **on warningLimit** und **on busOff** werden beim Übergang des Zustands oder bei einer Änderung der Fehlerzähler des CAN-Controllers aufgerufen. Verwenden Sie diese Prozeduren, um die Fehlerzähler zu überwachen (z.B. Ausgabe einer Warnung), um die Messung ggf. zu beenden oder um nach einem Übergang in den Busoff-Zustand einen Reset durchzuführen.

Innerhalb der Ereignisprozeduren des CAN-Controllers stehen die Fehlerzähler des CAN-Controllers über das Schlüsselwort **this** zur Verfügung:

```
on errorPassive {
    ...
    write("CAN Controller ist in errorPassive")
    write("    errorCountTX = %d", this.errorCountTX);
    write("    errorCountRX = %d", this.errorCountRX);
};
```

Hinweis: Der Zugriff auf die Werte der Fehlerzähler wird bislang nur von Systemen mit der CANcardX unterstützt.

3.2.4 Ausdrücke in CAPL

Die CAPL Syntax sind an die Programmiersprache C angelehnt. Folgende Ausdrücke sind wie in C erlaubt:

- Anweisungsblöcke: **{ ... }**
- **if { ... }** und **if {...} else { ... }**
- **switch**, **case**, **default**
- **for..**, **while..**, **do..while**-Schleifen
- **continue** und **break**
- **return**
- **?:** Ausdruck

Beispiele zu diesen Ausdrücken finden Sie in der Online-Hilfe des CAPL-Browsers

In CAPL stehen Ihnen die gleichen arithmetischen, logischen und bitweisen Operatoren (**+**,*****, **+=**, ***=**, **||**, **++**, ...) wie in C zur Verfügung.

Mit der Cast-Anweisung kann (wie in C) eine Typumwandlung explizit vorgenommen werden. Der Cast mit Angabe einer **message**-Denotation kann verwendet werden, um Botschaften mit verschiedenen Identifiern zuzuweisen.

Bei der Auswertung von Ausdrücken die **long**, **dword**, **int**, **word**, **char** und **byte** enthalten, erfolgt eine implizite Typumwandlung. Es gelten folgende Regeln:

8 Bit Größen werden vorher nach 32 Bit gewandelt (**char** α **long**, **byte** α **dword**)

16 Bit Größen werden vorher nach 32 Bit gewandelt (**int** α **long**, **word** α **dword**)

3.2.5 Das Schlüsselwort this

Mit dem Schlüsselwort **this** wird innerhalb einer Ereignisprozedur für den Empfang eines CAN-Objekts oder einer Umgebungsvariable die Datenstruktur dieses Objektes bezeichnet.

Beispielsweise greifen Sie mit

```
on message 100 {
   byte byte_0;
   byte_0 = this.byte(0);
   ...
}
```

auf das erste Datenbyte der Botschaft 100 zu, die gerade empfangen wird. Analog lesen Sie in CANoe mit

```
on envVar Schalter {
   int val;
   val = getValue(this);
   ...
}
```

den neuen Wert der gerade geänderten Ganzzahl-Umgebungsvariable *Schalter* aus.

Sie sollten den Wert von **this** innerhalb einer Ereignisprozedur nicht verändern. Damit Sie **this** aber als Parameter verwenden können, werden Werteänderungen von **this** vom CAPL-Compiler nicht verboten. Beachten Sie jedoch, daß derartige Schreibzugriffe auf **this** nur lokal (d.h. innerhalb der Ereignisprozedur) gültig sind. Bei Compilieren erhalten Sie eine entsprechende Warnung. Wenn Sie also nach einer Änderung von this in einer **on message**-Ereignisprozedur die Funktion **output(this)** aufrufen, wird das unveränderte Original von **this** und nicht Ihre Änderung weitergeleitet.

3.2.6 Interpretation von Signalwerten

Bei Verwendung einer Projektdatenbasis kann auf die einzelnen Signale durch Angabe ihres Namens zugegriffen werden. Der Compiler übernimmt die Bestimmung des Bit-Offsets und der Bitlänge sowie die Umsetzung des Maschinenformats (Intel bzw. Motorola). Die Werte werden im Rohformat interpretiert. Soll die physikalische Interpretation verwendet werden, so muß der Operator **phys** angehängt werden. Der

Compiler skaliert den Wert dann nach der in der Datenbasis angegebenen Umrechnungsformel (Faktor, Offset).

Das folgende Beispiel geht davon aus, daß Sie in der zugeordneten Datenbasis die Botschaft *Motordaten* mit einem Signal *Drehzahl* definiert haben und daß Sie ferner eine Umrechnungsformel angegeben haben.

```
message Motordaten m;
char    errSymbol[32];
...
m.Drehzahl=100;        // Zuweisung eines Rohwertes
m.Drehzahl.phys=123.5; // Anwendung d. Umrechnungsformel
```

In der ersten Zeile wird der Rohwert 100 den Datenbits zugewiesen, die Sie in der Datenbasis dem Signal Drehzahl zugeordnet haben. In der zweiten Zeile wird zur Berechnung des Bitwerts die in der Datenbasis zugeordnete Umrechnungsformel herangezogen.

Beachten Sie beim Arbeiten mit Signalen, daß die Speicherung der Signalwerte immer diskret erfolgt. Wird ein physikalischer Wert zugewiesen, so wird nach der Skalierung der nächstliegende diskrete Rohwert eingetragen. Wird das Signal danach ausgelesen, so ist das Ergebnis nicht zwangsläufig der Ursprungswert.

Nehmen Sie beispielsweise an, daß das Signal *Drehzahl* der Botschaft *Motordaten* definiert sei als 16 Bit unsigned mit einem Offset von 0 und einem Faktor von 10. Wird als physikalischer Wert 1015 zugewiesen, müßte der Rohwert 1015/10 = 101.5 sein. Da nur diskrete Werte vorkommen, wird 101 gespeichert. Beim Auslesen wird dieser Wert zu 101*10 = 1010 skaliert.

3.2.7 Zugriff auf Attribute in CAPL

Neben den Datenbytes, die mit einer CAN-Botschaft über den Bus gesendet werden, sind in bestimmten Systemen zusätzliche Attribute für Botschaften, Signale oder Netzknoten von Interesse. Beispielsweise kann man mit Attributen die Zykluszeiten aller zyklische gesendeter Botschaften einen CAN-Systems beschreiben. Man definiert dazu ein Integer-Attribut *Zykluszeit [in ms]* für alle Botschaften und legt dann für alle zyklische zu sendenden Botschaften einen Wert für dieses Attribut fest.

Die CAN-Datenbasis unterstützt Attribute für folgende Objektklassen:

- Botschaften
- Signale
- Netzknoten
- Datenbasis
- Umgebungsvariablen (für CANoe)

Attribute werden beschrieben durch einen Attributname, einen Datentyp, einen Wertebereich und einen Default-Wert. Mögliche Datentypen sind INTEGER, HEX, FLOAT, ENUM und STRING. Der Wertebereich für den Typ ENUM wird durch eine Aufzählung von gültigen Namen angegeben, sonst durch Minimum und Maximum.

Sie können jedem Attribut eines jeden Datenbasis-Objekts Werte zuweisen. Die Attribute aller Objekte, denen Sie keine Attributwerte zugewiesen haben, werden mit den Defaultwerten belegt.

In CAPL selektieren Sie den Attributwert eines Datenbasis-Objekts, indem Sie einen Punkt (**.**) zwischen Datenbasis-Objekt und Attributnamen schreiben. Beispielsweise werden in dem CAPL-Programmausschnitt

```
float rpm;
int time, n;
char buff[100];
...
time = Motordaten.Zykluszeit;
n = ABS.Knotennummer;
rpm = Motordaten.Geschwindigkeit.Max_rpm;
strncpy(buff,motordat.db_name,elcount(buff)-1);
```

die Werte des Botschaftsattributs **Zykluszeit** (Objekt: Botschaft **Motordaten**, Typ: INT), des Signalattributs **Max_rpm** (Objekt: Signal **Geschwindigkeit** der Botschaft **Motordaten**, Typ: FLOAT), des Netzknotenattributs **Knotennummer** (Objekt: Netzknoten **ABS**, Typ: INT) und des Datenbasisattributs **db_name** (Objekt: Datenbasis **motordat**, Typ: STRING) ausgelesen.

Attributwerte vom Typ ENUM werden in CAPL wie Strings behandelt. Mit der letzten Zeile des obigen Beispiels kopieren Sie also den Wert des ENUM-Botschaftsattributs in den Puffer **buff**. Beachten Sie, daß alle Attribute den gleichen Namensraum besitzen. Es ist daher z.B. nicht möglich, ein Datenbasisattribut und ein Botschaftsattribut gleichen Namens gleichzeitig zu verwenden.

Hinweis: Falls in CAPL den Attributwert eines Objektes auslesen, ohne daß Sie diesen Wert für das Objekt explizit in der Datenbasis gesetzt haben, erhalten Sie den Default-Wert, den Sie für jedes Attribut bei der Definition im Datenbasiseditor angeben.

3.2.8 Arbeiten mit symbolischen Signalwerten

Neben symbolischen Namen für Botschaften, Signale und Umgebungsvariablen können Sie auch symbolische Bezeichner für Ganzzahlwerte von Signalen und Umgebungsvariablen verwenden. Ordnen Sie dazu im Datenbasiseditor den entsprechenden Ganzzahlwerten von Signalen und Umgebungsvariablen symbolische Namen zu.

In CAPL verwenden Sie anstatt der Werte diese Namen, indem Sie den zugehörigen Signal- bzw. Umgebungsvariablennamen gefolgt von zwei Doppelpunkten (**::**) voranstellen. Wenn Sie also in der Datenbasis `AMPEL.CFG` die Werte des Signals *AmpelStatus* der Botschaft *Ampeldaten* folgendermaßen definiert haben

Wert	1	2	3
Name	Rot	Gelb	Grün

dann schreiben Sie in CAPL folgenden Code, um eine Botschaft *Ampeldaten* mit dem symbolischen Wert *Rot* (=1) des Signals *AmpelStatus* auf den Bus auszugeben.

```
message Ampeldaten msg;
msg.AmpelStatus = Ampeldaten.AmpelStatus::Rot;
output(msg);
```

Wenn Sie den symbolischen Bezeichner eines Signalwerts lesen möchten, verwenden Sie den Operator **sys**:

```
on message Ampeldaten
{
   char buf[64];
   strncpy(buf,this.AmpelStatus.sym,elCount(buf)-1);
   write("this.AmpelStatus = %s",buf);
}
```

Analog können Sie auch bei Umgebungsvariablen von Typ Integer mit symbolischen Werten arbeiten, wenn Sie diese in der Datenbasis definiert haben.

3.2.9 Arbeiten mit mehreren Datenbasen

Bei großen Systemen ist es u.U. sinnvoll, die Beschreibung der Botschaften und Signale sowie die Umgebungsvariablen auf mehrere Teildatenbasen aufzuteilen. Auch bei der Analyse von zwei CAN-Systemen gleichzeitig liegt es nahe, jedes System durch eine eigene Datenbasis zu beschreiben.

CAPL unterstützt die gleichzeitige Verwendung mehrerer Datenbasen. Im Browser konfigurieren Sie die Datenbasen und den dieser Datenbasis zugeordneten CAN-Chip mit dem Menüpunkt *Datei|Datenbasis* zuordnen. Sie können danach die symbolischen Bezeichner für Botschaften, Signale und Umgebungsvariablen aus allen Datenbasen verwenden.

Hinweis: Wenn Sie den Browser aus dem CANalyzer bzw. aus CANoe heraus starten, etwa indem Sie auf einen CAPL-Knoten doppelklicken, werden die im CANalyzer bzw. in CANoe konfigurierten Datenbasen automatisch richtig zugeordnet.

Falls der symbolische Botschaftsname *MotorDaten* in einer Datenbasis steht, die explizit dem Controller CAN2 zugeordnet ist, so wird mit

```
message MotorDaten msg; // MotorDaten ⇔ Chip2
...
output(msg); // msg wird ueber Chip 2 gesendet
```

die Botschaft **msg** über Chip 2 gesendet. Wenn Sie die Controller-Zuordnung der Datenbasis auf CAN1 ändern, wird nach einem erneutem Compilieren beim Durchlaufen des selben CAPL-Codes die Botschaft über Chip 1 gesendet.

Bei der Verwendung von mehreren Datenbasen können Sie mehrdeutige Symbole auflösen, indem Sie diese qualifizieren. Schreiben Sie zu diesem Zweck einfach den Datenbasisnamen gefolgt von zwei Doppelpunkten (**::**) vor den symbolischen

Namen. Nehmen Sie beispielsweise an, in der Datenbasis *Infodat* sei die Botschaft 0x10F mit *DiagData* bezeichnet. In der Datenbasis *Motordat* dagegen werde der Name *DiagData* verwendet als Name für die Botschaft 0x4E1. Mit **`Infodat::DiagData`** für die Botschaft 0x10F und mit **`Motordat::DiagData`** für die Botschaft 0x4E1 erhalten Sie eindeutige Bezeichner für beide Botschaften.

Weitere Hinweise über die Auflösung von Mehrdeutigkeiten beim Arbeiten mit mehreren Datenbasen finden Sie in der Online-Hilfe.

3.2.10 Funktionen in CAPL

CAPL-Funktionen dienen u.a. zum Kapseln von Code zur Wiederverwendung sowie zur Schnittstellenbildung. Die Syntax ist C-ähnlich, hat aber im Gegensatz zu C eine Reihe zusätzlicher Features:

- Ein fehlender Ergebnistyps wird als void interpretiert
- Eine leere Parameterliste ist wie in C++ erlaubt
- Überladen von Funktionen (d.h. mehrere Funktionen mit dem gleichen Namen, aber unterschiedlichen Parameterlisten) ist wie in C++ möglich
- Eine Parameterprüfung erfolgt wie in C++
- Arrays beliebiger Dimension und Größe dürfen übergeben werden

Beispielsweise gibt folgende Funktion eine Matrix ins Write-Fenster aus:

```
void write(int m[][])
{
   int i,j;
   for(i=0;i<elCount(m);i++) {
      for(j=0;j<elCount(m[0]){
         write("%ld", m[i][j]);
      }
   }
}
```

Beachten Sie, daß es bereits eine intrinsische CAPL-Funktion **`write()`** gibt, mit der Sie Text formatiert ins Write-Fenster ausgeben können. Die im Beispiel definierte Funktion überlädt also die intrinsische Funktion und wird statt dieser immer dann aufgerufen, wenn ihr ein zweidimensionales Feld übergeben wird.

Eine vollständige Bibliothek der intrinsischen CAPL-Funktionen finden Sie in der Online-Hilfe des CAPL-Browsers. Die folgende Tabelle bietet eine kurze Übersicht über die wichtigsten Funktionen.

Senden von Frames	**`output(), output(errorFrame)`**
Ausgabe ins Write-Fenster	**`write()`**
Ausgabe in Logging-Datei	**`writeToLog(), writeToLogEx()`**
Messungskontrolle	**`trigger(), stop()`**
Zeitverwaltung	**`setTimer(), cancelTimer(), timeNow()`**

Feldgrößenbestimmung	**elCount()**
Umgebungsvariablenzugriff	**getValue(), putValue()**
PC Zugriff	**outport(), inport()**
Berechnungen	**sin(), cos(), abs()**

3.3 Erstellen eines CAPL-Programms

Nehmen Sie an, Sie wollen ein CAPL-Programm erstellen, mit dem Sie Botschaften mit dem Identifier 0x100 zählen und die Anzahl auf Tastendruck im Write-Fenster ausgeben können. Wie gehen Sie vor?

Zunächst müssen Sie entscheiden, an welcher Stelle Sie das CAPL-Programm im Datenflußplan einfügen. Prinzipiell steht Ihnen dazu jeder Hot-Spot im Meßaufbau und in CANoe zusätzlich der Simulationsaufbau zu Verfügung. Da es sich um ein reines Analyse-Programm handelt, das selbst keine Botschaften erzeugt und auf den Bus legt, bietet es sich an, das Programm rechts im Meßaufbau, etwa vor dem Statistikfenster einzufügen. Wählen Sie im Kontextmenü des Hot-Spots die Funktion *Füge CAPL-Knoten ein.* Im Meßaufbau erscheint an der gewählten Stelle nun ein Funktionsblock mit dem Programmsymbol *P*. Den Konfigurationsdialog dieses Knotens erreichen Sie ebenfalls über das Kontextmenü. Wählen Sie hier zunächst einen Programmnamen, z.B. `COUNTER.CAN`, und starten Sie dann den Browser entweder über die Schaltfläche *Edit...* des Konfigurationsdialoges oder direkt mit einem Doppelklick auf den Programmblock im Meßaufbau.

Sofern Sie eine neue Datei gewählt haben, öffnet sich der in Abschnitt 3.4 beschriebene Browser mit seinen 4 Teilfenstern („Panes“), in denen die (noch) leere Programmdatei dargestellt wird.

Zunächst benötigen Sie für das Programm eine Ganzzahlvariable, welche die Botschaften zählt. Nennen Sie sie z.B. **counter**. Gehen Sie in die rechte obere Pane, und tragen Sie diesen Namen im Variablenblock ein. In diesem Teilfenster sollte nun

```
variables {
   int counter;
}
```

stehen. Die Variable wird - wie alle globalen Variablen - bei Messungsstart automatisch auf Null initialisiert.

Im nächsten Schritt soll diese Variable inkrementiert werden, immer wenn eine Botschaft mit Id 0x100 registriert wird. Sie müssen das CAPL-Programm also um eine Ereignisprozedur vom Typ **on message** erweitern. Klicken Sie dazu in der linken oberen Pane den Ereignistyp **Message** an und fügen Sie in der linken unteren Pane mit dem Kontextmenü eine neue Ereignisprozedur dieses Typs hinzu.

In der rechten unteren Pane erscheint nun ein Prozedurrumpf. Ersetzen Sie hier zunächst den Text **<newMessage>** durch den Identifier **0x100**. Jetzt bleibt nur noch zu definieren, welche Aktionen bei Eintreten des Ereignisses ausgeführt werden

sollen: Da das Programm Botschaften zählen soll, muß die Variable **counter** beim Registrieren einer Botschaft inkrementiert werden. Die fertige Prozedur sind folgendermaßen aus:

```
on message 0x100
{
   counter++;
}
```

Im letzten Schritt muß nun noch die Ausgabe ins Write-Fenster implementiert werden. Schließlich soll das Programm nicht nur Botschaften zählen, sondern Sie möchten auch kontrollieren, wie viele Botschaften gezählt worden sind.

Die Ausgabe soll auf Drücken der Taste ‘a’ ins Write-Fenster erfolgen. Sie müssen also eine weitere Ereignisprozedur für das Ereignis „Drücke Taste ‘a‘“ definieren. In der linken oberen Pane wählen Sie dazu den Typ **Keyboard**. Die vorher definierte **on message**-Prozedur wird damit unsichtbar, da sie einem anderen Ereignistyp angehört. Sie bleibt natürlich weiterhin Bestandteil des CAPL-Programms und wird wieder sichtbar, sobald Sie erneut den Ereignistyp **Message** anwählen.

Fügen Sie nun dem CAPL-Programm in der unteren linken Pane über das Kontextmenü eine **Keyboard**-Ereignisprozedur hinzu. In der rechten unteren Pane erscheint ein neuer Prozedurrumpf, den Sie folgendermaßen ausfüllen:

```
on key ’a’
{
   write("%d Botschaften 0x100 gezaehlt",counter);
}
```

Die Formatangabe %d bezieht sich auf die Ganzzahlvariable **counter**, die hinter dem Komma angegeben ist. Der Format-String entspricht weitgehend dem der C-Funktion `printf()`.

Das Programm ist damit fertig. Speichern Sie es ab und starten Sie dann den Compiler mit der F9-Taste, über das Hauptmenü *Compiler|Compilieren* oder über den Blitz in der die Werkzeugleiste. Falls Sie beim Erstellen des Programmes einen Fehler gemacht haben, öffnet sich ein Meldungsfenster, das Ihnen den Fehler anzeigt. Doppelklicken Sie auf diese Fehlermeldung, um an die Stelle zu gelangen, an der der Fehler aufgetreten ist. Nachdem Sie ihn korrigiert und die Programmdatei erneut abgespeichert haben, compilieren Sie das Programm erneut. Wenn das Programm fehlerfrei compiliert werden konnte, erscheint in der Statuszeile unten im Hauptfenster des Browsers die Meldung *compiled*.

Starten Sie nun die Messung. Immer wenn Sie die Taste ‘a’ drücken, werden Sie nun den Text „*n* Botschaften gezaehlt“ im Write-Fenster sehen, wobei *n* immer die Anzahl der gezählten Botschaften bezeichnet.

3.4 Browser zum Erstellen und Compilieren der CAPL-Programme

CAPL-Programmdateien sind genauso wie C- oder PASCAL-Programmdateien ASCII-Files, können also mit jedem ASCII-Texteditor bearbeitet werden.

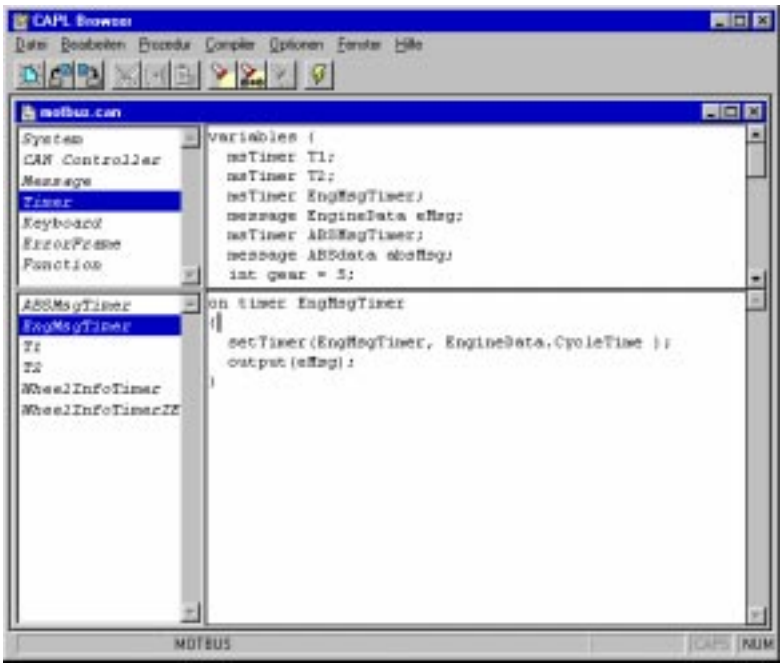


Abbildung 63: CAPL Browser

Zur komfortablen Erstellung und Änderung von CAPL-Programmen ist in den CANalyzer bzw. in CANoe ein spezieller Browser integriert (engl. browse = schmökern), der Ihnen die Variablen, Ereignisprozeduren und Funktionen eines CAPL-Programms in strukturierter Form anzeigt.

Es können gleichzeitig mehrere Browser-Fenster mit unterschiedlichen CAPL-Programmen geöffnet sein.

Der CAPL-Compiler wird im Browser über das Hauptmenü oder über die Werkzeugleiste gestartet. Die Compilationszeit ist auch bei größeren Programmen sehr gering. Wird ein Fehler erkannt, so wird der fehlerhafte Programmteil dargestellt und der Cursor wird auf die Fehlerstelle positioniert. Korrekturen können somit sehr einfach ausgeführt werden.

3.4.1 Öffnen des Browsers

Der Browser befindet sich im Systemverzeichnis (canbrow.exe für die 16-bit Version bzw. canbr32.exe für die 32-bit Version) und kann als eigenständige Applikation so wie jedes andere Windows-Programm gestartet werden. Dennoch empfiehlt es sich, den Browser stets über den CANalyzer bzw. über CANoe zu starten, da für den Start des Programmes eine ganze Reihe wichtiger Parameter (Datenbasisname, Compiler-Optionen, Hardwareparameter, CAPL-DLLs, ...) übergeben werden muß.

Wenn Sie den Browser durch Doppelklicken auf den CAPL-Knoten im Meß- bzw. Simulationsaufbau starten, wird automatisch die Datei PARBROW.INI mit allen Startparametern generiert und dem Browser zur Verfügung gestellt. Starten Sie den Browser daher stets über den CANalyzer bzw. über CANoe und achten Sie ferner darauf, die Datei PARBROW.INI niemals mit einem Schreibschutz zu versehen, da sonst beim nächsten Start des Browsers, wichtige Startparameter u.U. nicht richtig übergeben werden können.

Hinweis: Eine CAPL Programmdatei kann aus dem Dateimanager auf einen bereits gestarteten CAPL-Browser gezogen werden („Drag and Drop“). Die Datei wird dann in einem neuen Fenster angezeigt.

3.4.2 Browser-Fenster

Ein Browser-Fenster ist in vier Teilfenster, sogenannte „Panes“, unterteilt. Auf der linken Seite befinden sich zwei Listen. Die obere zeigt Ihnen die CAPL-Ereignistypen, die untere enthält die Namen der CAPL-Prozeduren, ist bei neuen Programmen also zunächst leer.

Die rechten Teilfenster enthalten Texteditoren mit den üblichen Funktionen zur Cursorbewegung, zum Suchen und Ersetzen und einer globalen Zwischenablage, mit dessen Hilfe Textstücke in andere Fenster bzw. Panes übertragen werden können.

Die Editor-Pane auf der oberen rechten Seite dient zur Deklaration der globalen Variablen des CAPL-Programms, in der darunterliegenden Pane können Sie den Programmtext für die in der Prozedurliste ausgewählte Prozedur editieren.

Die wichtigsten Funktionen erhalten Sie für jede der 4 Panes über das Kontextmenü mit der rechten Maustaste. In den Editor-Panes haben Sie über das Kontextmenü Zugriff auf die intrinsischen CAPL-Funktionen und auf die in der Datenbasis definierten Objekte. Ferner können Sie mit dem Kontextmenü Text in die Zwischenablage kopieren und von dort aus in Ihr Programm einfügen.

CAPL-Programme, die nicht im Browser-spezifischen Dateiformat vorliegen, werden in unstrukturierter Form in einem normalen Textfenster angezeigt und können dort bearbeitet werden. Wie in den Editoren des Browser-Fensters kann der Programmtext mit den Kommandos des Menüs *Bearbeiten* bzw. über das Kontextmenü editiert werden. Wie Sie solche CAPL-Programme im Browser-Format importieren können, erfahren Sie in Abschnitt 3.4.6.

3.4.3 Compilieren von CAPL-Programmen

Um aus einem CAPL-Programm eine ablauffähige Programmdatei im CBF-Format (CAPL Binary Format, Dateiendung *.CBF) zu erzeugen, müssen Sie das Programm mit dem CAPL-Compiler übersetzen.

Das Compilieren eines Programms wird durch das Kommando *Compilieren* aus dem Compilermenü, durch Drücken des Compile-Symbols ( ) in der Werkzeugleiste oder durch Betätigen der F9-Taste gestartet. Falls das CAPL-Programm noch keinen Dateinamen besitzt, werden Sie aufgefordert, einen Namen für die CBF-Datei einzugeben. Besitzt das Programm bereits einen Namen, setzt sich der Name der CBF-Datei aus dem Programmnamen und der Endung CBF zusammen.

Alle Meldungen während des Compiliervorgangs werden im Meldungsfenster ausgegeben, das Sie über den Hauptmenüpunkt *Fenster|Meldungen* öffnen können.

Falls während des Compiliervorgangs Fehler oder Warnungen auftreten, springt das Meldungsfenster mit dem entsprechenden Hinweis von selber in den Vordergrund. Doppelklicken Sie auf die Fehlermeldung oder selektieren Sie die Zeile und führen Sie das Kommando *Gehe zu* aus dem Menü *Meldungen* aus, um den Cursor an die Stelle zu positionieren, an der der Fehler aufgetreten ist. Nachdem Sie ihn korrigiert und die Programmdatei erneut abgespeichert haben, compilieren Sie das Programm erneut. Wenn das Programm fehlerfrei compiliert werden konnte, erscheint in der Statuszeile unten im Hauptfenster des Browsers die Meldung *compiled*.

3.4.4 Suchen von Laufzeitfehlern

Schwieriger als die Suche von Syntaxfehlern, die bereits zur Complierzeit gefunden werden, ist meist die Suche nach Laufzeitfehlern. Dennoch bietet CAPL Ihnen hier in zweifacher Hinsicht Unterstützung an.

Einerseits werden eine Reihe von möglichen Laufzeitfehlern in CAPL-Programmen automatisch überwacht. Hierzu gehören

- eine Division durch Null
- ein Über- oder Unterschreiten von Array-Grenzen
- ein Über- oder Unterschreiten von Offsets in den Datenbereich von Botschaften
- ein Stackoverflow beim Aufruf von CAPL-Unterprogrammen

Wird solch ein Laufzeitfehler entdeckt, wird die Messung kontrolliert beendet. Sie erhalten einen Hinweis im Write-Fenster mit den Namen des CAPL-Programms, der Fehlerart und einem Fehlerindex enthält. Mit Hilfe diese Fehlerindex können Sie die Stelle, die den Fehler verursacht hat, leicht im CAPL-Quelltext finden: geben Sie den Index unter dem Menüpunkt *Compiler|Laufzeitfehler* suchen ein.

Andererseits habe Sie als Anwender die Möglichkeit, mit der CAPL-Funktion **`runError()`** selber Laufzeitfehler zu generieren und können so gezielt kritische Programmstellen absichern. Sie können der Funktion dabei wiederum einen Fehlerindex mitgeben, der dann beim Messungsabbruch im Write-Fenster ausgegeben wird.

3.4.5 Zugriff auf die Datenbasis

Sofern Sie den Browser direkt aus dem Meß- bzw. Simulationsaufbau heraus öffnen, erhält der Browser die Information über die der Konfigurationzugeordneten Datenbasen über die temporäre Datei `PARBROW.INI`. Sie können dem CAPL-Programm aber auch im Browser direkt eine oder mehrere Datenbasen über das Dateimenü zuordnen.

Im Browser haben Sie über den Botschafts- und Signal-Explorer Zugriff auf die Botschaften und die Signale der Datenbasis. Sie öffnen diesen Auswahldialog über den Kontextmenüeintrag *CANdb Botschaft* bzw. *CANdb Signal* der rechten Editor-Panes. Das selektierte Objekt wird nach Verlassen des Dialoges mit *OK* direkt in den CAPL-Programmtext übernommen. Statt über den symbolischen Auswahldialog,

können Sie Signal- und Botschaftsnamen in den Editor-Panes natürlich auch direkt als Klartext eingeben.

Wenn Sie mehr als eine Datenbasis verwenden, werden in den Symbolauswahldialogen die Objekte in den auf die erste folgenden Datenbasen mit dem Datenbasisnamen qualifiziert. Sie benötigen diese qualifizierten Namen jedoch nur, um Mehrdeutigkeiten aufzulösen. Solange die symbolischen Namen in allen Datenbasen eindeutig sind, können Sie beim Editieren von CAPL-Programmen auf die Qualifizierung der symbolischen Namen verzichten.

Auf was Sie bei der Verwendung mehrerer Datenbasen achten sollten, können Sie in Abschnitt 4.3.2 nachlesen.

3.4.6 Importieren und Exportieren von ASCII-Dateien

Um Ihre CAPL-Programme außer mit dem Browser auch mit anderen Texteditoren bearbeiten zu können, bietet Ihnen der Browser eine Import- und eine Export-Funktion im Dateimenü.

Die Strukturinformationen, die der Browser verwendet, um Ihnen CAPL-Programme im Browser-Format anzuzeigen, befinden sich in speziellen Kommentarzeilen. Der Browser generiert diese Informationen beim Compilieren automatisch.

Mit *Import* kann ein CAPL-Programm, das nur im ASCII-Format vorliegt in den CAPL-Browser geladen werden. Die im Programm definierten Prozeduren werden den CAPL-Prozedurgruppen (z.B. Message, Timer, CAPL-Funktion, ...) zugeordnet. Das Kommando öffnet einen Dateiauswahldialog mit dem Sie die zu importierende Datei auswählen. Achten Sie beim Importieren von Programmen darauf, daß die Programme fehlerfrei compiliert werden können, da sonst keine eindeutige Erkennung der Funktions- bzw. Prozedurnamen erfolgen kann.

Mit *Export* können Sie CAPL-Programme im ASCII-Format abspeichern. Die erzeugte Datei enthält keinerlei Strukturinformation des CAPL-Browsers und kann nur durch ein späteres Importieren in einem Browser-Fenster angezeigt werden.

3.4.7 Browser-Optionen

Im Menü *Optionen|Editor...* finden Sie Kommandos für die individuelle Einstellung der Editoren der Browser-Panes

Durch Drücken der Knöpfe im Schriftenfeld können Sie die Bildschirm bzw. Druckerschriften auswählen. Die Auswahl erfolgt über einen Schriftenauswahldialog

Im darunterliegenden Textfeld wird die Breite einer Tabulatorposition eingegeben.

Die Änderungen der Einstellungen werden nach Betätigen von *OK* wirksam, danach werden alle geöffneten Editoren an die neuen Einstellungen angepaßt. Mit *Abbrechen* werden die Einstellungsänderungen verworfen.

Der Menüpunkt *Optionen|Compiler...* öffnet einen Dialog in dem die Compiler-Optionen für das aktuelle CAPL-Programm geändert werden können. Sofern Sie den Browser über den Meßaufbau oder den Simulationsaufbau geöffnet haben, werden diese Optionen automatisch für Sie eingestellt. Sie sollten die Einstellungen hier also normalerweise nicht verändern.

3.5 CAPL Sprachbeschreibung

In diesem Kapitel werden die Syntax der Sprachelemente in einer erweiterten BNF-Darstellung und die Semantik in Form von erläuterndem Text beschrieben.

Die syntaktische Konstrukte wurden, soweit vernünftig möglich, an die Programmiersprache C angelehnt.

3.5.1 Erweiterte Backus-Naur-Form

Zur Beschreibung der Grammatik der Sprache wird eine erweiterte Backus-Naur-Form (BNF) verwendet:

- Ein Nonterminal wird als einfacher Bezeichner dargestellt:

```
varDeclList   msgDenotation
```

- Als Ableitungssymbol in den Produktionsregeln wird '::=' verwendet.
- Ein Terminal (Schlüsselwort) wird in Anführungszeichen eingeschlossen. Bei der Programmeingabe wird Groß- und Kleinschreibung bei Schlüsselwörtern nicht unterschieden:

```
"VARIABLES"   "TIMER"   "{"
```

- Nicht formal definierte Teile werden in Spitzklammern eingeschlossen:

```
<integer>    <varId>
```

- Ein optionaler Teil wird in eckige Klammern eingeschlossen:

```
varDecl [ varDeclList ]
```

- Alternative Teile werden durch '|' getrennt:

```
"WORD"  |  "INT"  |  "BYTE"  | "CHAR"
```

- Erläuterungen sind in Kommentarzeichen eingeschlossen

```
/* unsigned 16 Bit */
```

- Es kann auch das C++ Kommentarzeichen // verwendet werden. Der Rest der Zeile dient dann als Kommentar

```
// Kommentarzeile
```

3.5.2 Programmaufbau

Ein CAPL-Programm besteht aus der Deklaration von (möglicherweise null) globalen Variablen und einer Menge von Event-Prozeduren (sinnvollerweise mindestens eine).

```
CAPLProgram      ::= varDeclarations procedures
```

3.5.3 Deklaration von Variablen

```
varDeclarations  ::=
                 "VARIABLES" "{"
                    [ varDeclList ]

                 "}"
```

Im Browser wird die Deklaration globaler Variablen in der Globalvariablen-Pane vorgenommen.

```
varDeclList       ::=
                  varDecl [ varDeclList ] |
                  constDeclaration [ varDeclList ] |

varDecl           ::=
                  intTypeSpecifier intIdList ";" |
                  "FLOAT" intIdList |
                  "DOUBLE" intIdList |
                  "TIMER" idList ";" |
                  "MSTIMER" idList ";" |
                  "MESSAGE" msgDenotation msgIdList ";"
                  "PG" msgDenotation msgIdList ";"

intTypeSpecifier  ::=
                  "DWORD" |           // unsigned 32 Bit
                  "LONG"  |           // signed   32 Bit
                  "WORD"  |           // unsigned 16 Bit
                  "INT"   |           // signed   16 Bit
                  "BYTE"  |           // unsigned  8 Bit
                  "CHAR"              // signed    8 Bit
```

Die Datentypen **dword**, **long**, **word**, **int**, **byte** und **char** können analog zu der Programmiersprache C verwendet werden.

Die Datentypen **float** und **double** sind Synonyme und bezeichnen 64 Bit Gleitkommazahlen nach dem IEEE Standard.

Beispiel:

```
int counter;
double speed, temperature;
```

Mit Variablen vom Typ **timer** und **msTimer** werden Timer erzeugt, die erst nachdem sie in einer Event-Prozedur gesetzt wurden, loslaufen. Bei Ablauf der Zeit wird die zugehörige Event-Prozedur aufgerufen. Auf diese Variablen kann nur über vordefinierte Funktionen zugegriffen werden. Bei **timer** wird die Zeit im Sekundentakt, bei **msTimer** im Millisekundentakt, erniedrigt. Aus Geschwindigkeitsgründen dient zur Speicherung eine 16 Bitgröße (intern zum Zählen der Ticks 32 Bit). Die Maximalzeit für **timer** beträgt 1799 s (Begrenzung durch interne Rechengenauigkeit), für **msTimer** 65,535 s (Begrenzung durch 16 Bit).

Beispiel:

```
timer timer1s;
msTimer timer100ms;
```

Mit **message** bzw. **pg** werden CAN-Objekte (Botschaften, J1939 Parameter-Groups) deklariert, die aus dem CAPL-Programm ausgegeben werden sollen. Der Zugriff auf die Komponenten der Objekte erfolgt über Selektoren (s.u.).

```
msgDenotation     :=
   [ chipSelector "." ] <integer> |      // Standard Id
   [ chipSelector "." ] <integer>"x" | // Extended Id
   [ chipSelector "." ] "*" |          // Wildcard
   [ chipSelector "." ] <msgSymbol>    // symbolischer Name
   [ chipSelector "." ] <dbName>":"<msgSymbol>
                     // symbolischer Name mit Datenbankqualifizierung
chipSelector      ::= "CAN1" | "CAN2"
```

Als Bezeichnung von CAN-Objekten wird die Botschaftskennung (Botschafts-Identifier) bzw. die Parameter-Group Number als Zahl (<integer>) in dezimaler oder hexadezimaler Darstellung angegeben. Für Extended Identifier wird ein **x** angehängt. Die Angabe von ***** bedeutet, daß diese Variable zunächst noch keine Botschaftskennung enthält. Diese muß dann in anderer Weise vor dem Versenden des Objektes festgelegt werden. Solche Objekte dienen z.B. bei Filteraufgaben zur Speicherung aller Objekte, die unverändert weitergereicht werden sollen. (Die empfangene Botschaft wird dann einschließlich des Botschafts-Identifiers kopiert.)

Bei Verwendung einer symbolischen Datenbasis kann statt der Zahl der entsprechende Botschaftsname aus der Datenbasis verwendet werden. Werden die symbolischen Bezeichner in mehr als einer Datenbasis definiert, kann der Botschaftsname mit dem Namen der Datenbasis qualifiziert werden, um Mehrdeutigkeiten zu vermeiden (angelehnt an die C++ Syntax).

Soll das Objekt (bei CAN-Karten mit mehr als einem CAN-Chip) für einen bestimmten Chip definiert werden, kann dies mit **CAN1** oder **CAN2** angegeben werden.

Beispiel:

```
message 100 msg100;
message Motordaten msgMotordaten;
message 6563x msgFirst, msgSecond;
message CAN1.ABSData msgABS;
message Motbus::Motordaten msgMotordaten;
```

```
idList                ::=    <varId> [ "," idList ]
intIdList             ::=    intIdDecl [ "," intIdList ]
intIdDecl             ::=    declarator[ "=" declInit ]
declarator            ::=    <varId> | declarator "[" <integer> "]"
declInit              ::=     "{" declInitList "}" |
                                        simpleType |
                                        <string>
declInitList          ::=    declInit [ "," declInitList ]
simpleType            ::=    signedInteger | signedFloat
signedInteger     ::= <integer> |
                                        "+" signedInteger | "-" signedInteger
signedFloat           ::=    <float> |
                                        "+" signedFloat | "-" signedFloat
```

Variablen können bei der Deklaration initialisiert werden. Hierfür ist sowohl die einfache Schreibweise als auch die Klammerung mit **{ }** möglich. Nicht explizit

initialisierte Variablen werden in CAPL auf 0 gesetzt. Die Deklaration von Feldern (Arrays, Vektoren, Matrizen) ist möglich. Sie werden analog zur Sprache C verwendet und initialisiert. Insbesondere können Felder vom Typ **`char`** durch die Zuweisung eines Strings initialisiert werden.

```
msgIdList            ::=      msgIdDecl [ "," msgIdList ]
msgIdDecl            ::=      msgDeclarator [ "=" msgDeclInit ]
msgDeclarator     ::=<varId> |
                                        msgDeclarator "[" <integer> "]"
msgDeclInit          ::=      "{" msgInitList "}" |
                                        "{" msgDeclInitList "}"
msgDeclInitList::=   msgDeclInit [ "," msgDeclInitList ]
msgInitList          ::=      msgInit [ "," msgInitList ]
msgInit              ::=      intTypeSpecifier
                              "(" <integer> ")" "=" signedInteger |
                              "DLC" "=" <integer> |
                              "RTR" "=" <integer> |
                              "DIR" "=" specialConstant |
                              "TYPE" "=" specialConstant |
                              "SIMULATED" "=" <integer> |
                              "CHARACTERISTIC" "=" <integer> |
                              "TIME" "=" <integer> |
                              <signalSymbol> "=" simpleType |
                              <signalSymbol>".PHYS" "=" simpleType |
                              "SA" "=" <integer> |
                              "DA" "=" <integer> |
                              "PRIO" "=" <integer> |
                              "R1" "=" <integer> |
                              "DP" "=" <integer>
```

Message-Variablen können bei der Deklaration initialisiert werden. Hierzu kann auf den Datenbereich durch Angabe des Typs und des Byte-Offsets zugegriffen werden. Für die Initialisierung der Kontrollbereiche sind symbolische Bezeichner vorgegeben.

Beispiel:

```
message 100 msg100 = {DLC=2, BYTE(0)=8, BYTE(1)=16 };
message Motordaten = { Speed.phys = 53.8 };
```

Es können auch Felder bzw. Matrizen von Message-Variablen angelegt werden.

Mit `specialConstant` werden symbolische Namen für numerische Konstanten bezeichnet. Ihre Verwendung ist mit Define-Konstanten in C vergleichbar.

Die Bedeutung der Kontrollinformation und der Konstanten wird weiter unten erläutert.

`<integer>`

Ganzzahlkonstante die als Dezimalzahl, als Hexadezimalzahl (mit 0x...) oder als Character-Konstante ('A') geschrieben werden kann. Bei der Zuweisung an einen Botschafts-Identifier wird ein X angehängt, wenn es sich um einen Extended-Identifier (29 Bit) handeln soll.

`<float>`

Gleitkommakonstante wie in C.

`<varId>`

Bezeichner für eine Variable. Muß mit einem Buchstaben oder '_' beginnen und kann dann "beliebig viele" Buchstaben, Ziffern und '_' enthalten. Groß-/Kleinschreibung ist relevant. Umlaute und 'ß' werden nicht als Buchstaben akzeptiert.

`<constant expression>`

Konstanter Ausdruck wie in C.

```
constDeclaration  ::=
                  "CONST" [ intTypeSpecifier | "DOUBLE" ] <varId> "="
                  <constant expression>
```

Hiermit werden Konstanten vereinbart. Dies entspricht den `#define`-Direktiven von C mit zusätzlicher Typinformation und Typprüfung. Wird kein Typ angegeben, wird long angenommen. Konstanten können überall dort verwendet werden, wo die Angabe von `<integer>` oder `<float>` vorgesehen ist.

3.5.4 Ereignisgesteuerte Prozeduren

Auf Ereignisse (Eintreffen einer Botschaft, Ablaufen eines Timers, ...) kann in ereignisgesteuerten Prozeduren (Event-Prozeduren) reagiert werden.

```
procedures          ::=     procedure [ procedures ]
procedure           ::=     eventProcedure | caplProcedure
eventProcedure    ::=
                  "ON" "MESSAGE" msgDenotation procBody    |
                  "ON" "TIMER" varId procBody                    |
                  "ON" "KEY" keyDenotation procBody              |
                  "ON" "PRESTART" procBody                       |
                  "ON" "START" procBody                          |
                  "ON" "STOPMEASUREMENT" procBody         |
                  "ON" "ERRORFRAME" procBody                     |
                  "ON" "ERRORACTIVE" procBody                    |
                  "ON" "ERRORPASSIVE" procBody                   |
                  "ON" "WARNINGLIMIT" procBody                   |
                  "ON" "BUSOFF" procBody
keyDenotation     ::= keyName | <integer> | "*"
```

```
keyName                 ::=
                    "F1" | "F2" | "F3" | "F4" | "F5" | "F6" |
                    "F7" | "F8" | "F9" | "F10" | "F11" | "F12" |
                    "shiftF1" | "shiftF2" | "shiftF3" | "shiftF4" |
                    "shiftF5" | "shiftF6" | "shiftF7" | "shiftF8" |
                    "shiftF9" | "shiftF10" | "shiftF11" | "shiftF12" |
                    "ctrlF1" | "ctrlF2" | "ctrlF3" | "ctrlF4" |
                    "ctrlF5" | "ctrlF6" | "ctrlF7" | "ctrlF8" |
                    "ctrlF9" |    "ctrlF10" |"ctrlF11" |"ctrlF12" |
                    "ctrlPageUp" |      "ctrlPageDown" |
                    "PageUp" | "PageDown" | "End" | "Home"
```

`msgDenotation` ermöglicht die gleichen Angaben wie bei der Variablendeklaration, wobei für jedes Objekt im gesamten Programm nur eine Event-Prozedur vereinbart werden darf. Die Angabe ***** bedeutet in diesem Zusammenhang, der Empfang einer Botschaft, für die keine eigene Event-Prozedur vereinbart wurde (Default-Event-Prozedur). Weiter gibt es die Möglichkeit, eine Liste von Botschaften durch Komma getrennt anzugeben. Auch Bereiche sind erlaubt.

Beispiel:

```
on message 100,200,250-300
on message Motordaten
on message *
on message Motbus::Motordaten
```

Der Code für einen Tastendruck (`keyDenotation`) kann entweder als Zahl oder mit Hilfe vordefinierter Bezeichnungen für Funktionstasten eingegeben werden. Die Angabe **`on key *`** dient als Wildcard. Diese Funktion erhält alle Tastendrücke.

Hinweis: Zahlen können auch als Character-Konstante, z.B. 'A', eingegeben werden. So ist für die "normalen" Tasten eine problemgerechte Eingabe gegeben.

Beispiel:

```
on key 65
on key ‘a’
on key pageDown
on key *
```

Für die Reaktion auf ein Timer-Ereignis muß die definierte Timer-Variable als Selektionskriterium angegeben werden.

Beispiel:

```
on timer Timer1sec
on timer Timer100ms
```

Der Prozedurkörper ist an die Möglichkeiten üblicher Programmiersprachen angelehnt.

```
procBody          ::=
                  "{"
                     [ localVarDecls ]
                     [ statements ]
                  "}"
localVarDecls     ::=  varDecl [ localVarDecls ]
```

Lokale Variablen werden in CAPL immer statisch angelegt (im Gegensatz zu C). Das bedeutet eine Initialisierung wird nur zum Programmstart durchgeführt und die Variablen besitzen beim Eintritt in die Prozedur den Wert, den sie beim letzten Verlassen der Prozedur hatten.

Die Deklaration von lokalen Variablen von Typ **`Timer`** ist nicht sinnvoll, da diese Event-Prozeduren triggern sollen. Diese können jedoch nur auf globale Timer reagieren.

```
statements        ::= statement [ statements ]

statement         ::= exprStatement        |
                                  compoundStatement    |
                                  selectionStatement   |
                                  iterationStatement   |
                                  jumpStatement        |
                                  caseStatement

compoundStatement ::= "{" statements "}"

selectionStatement   ::=
                  "IF" "(" expr ")"statement [ "ELSE" statement ] |
                  "SWITCH" "(" expr ")" statement

caseStatement           ::=
                  "CASE" signedInteger ":" statement | "DEFAULT" ":"

iterationStatement   ::=
                  "WHILE" "(" expr ")" statement                |
                  "DO" statement "WHILE" "(" expr ")" ";"  |
                  "FOR" "("[ expr ]";"[ expr ]";"[ expr ]")" statement

jumpStatement     ::=
                  "CONTINUE"          |
                  "BREAK"             |
                  "RETURN" [ expr ]
```

Die zur Verfügung stehende Anweisungsarten sind aus C übernommen und besitzen die gleiche Syntax und Semantik. Zur Vereinfachung von CAPL wurde auf **`goto`** und der Möglichkeit zur Deklaration von Variablen nach **`{`** verzichtet. Wie in C hat der Programmierer darauf zu achten, keine Endlosschleifen zu programmieren.

```
exprStatement     ::= [ expr ] ";"

expr              ::= leftValue "=" expr     |
                      leftValue "|=" expr           |
                      leftValue "&=" expr           |
                      leftValue "+=" expr           |
                      leftValue "%=" expr           |
                      leftValue "*=" expr           |
                      leftValue "-=" expr           |
                      leftValue "/=" expr           |
                      leftValue "<<=" expr    |
                      leftValue ">>=" expr    |
                      leftValue "^=" expr           |
                      dyadicOperation         |
                      dyadicOperation "?" expr ":" expr

leftValue ::= unaryExpr

dyadicOperation   ::=
                  dyadicOperation [ dyadicOperator castExpr ]

dyadicOperator    ::=
                  "||" |                          // conditional or
                  "&&" |                          // conditional and
                  "|"  |                          // bitwise inclusiv or
                  "^"  |                          // bitwise exlusiv or
                  "&"  |                          // bitwise and
                  "==" | "!=  |       // equality operators
                  ">=" | "<=" | ">" | "<"  | // relational operators
                  ">>" | "<<" |       // shift operators
                  "+"  | "-"  |       // add, sub
                  "*"  | "/"  |       // mul, div
                  "%"  |                          // modulo
```

Die dyadischen Operatoren entsprechen der Syntax von C. In der obigen Regel sind in einer Zeile die Operatoren mit der jeweils gleichen Präzedenz dargestellt. Die Präzedenz nimmt von oben nach unten zu.

Bei den Shift-Operatoren ist der Wertebereich zu beachten:

0 <= Shift-Wert <= 31.

Dies wird aus Laufzeitgründen nicht vom compilierten Code überwacht.

```
castExpr          ::=
                  "(" typeName ")" castExpr  | unaryExpr

typeName          ::=
                  "intTypeSpecifier"                              |
                  "FLOAT" | "DOUBLE"                              |
                  "MESSAGE" msgDenotation                         |
                  "PG" msgDenotation                      | (nur J1939)
```

Mit der Cast-Anweisung kann (wie in C) eine Typumwandlung explizit vorgenommen werden. Der Cast mit Angabe einer `msgDenotation` kann verwendet werden, um Botschaften mit verschiedenen Identifiern zuzuweisen. Die Verwendung der symbolischen Notation wird unten erläutert.

Bei der Auswertung von Ausdrücken die **`long`**, **`dword`**, **`int`**, **`word`**, **`char`** und **`byte`** enthalten, erfolgt eine implizite Typumwandlung. Es gelten folgende Regeln:

- 8 Bit Größen werden vorher nach 32 Bit gewandelt (**`char`**⇒**`long`**, **`byte`**⇒**`dword`**)
- 16 Bit Größen werden vorher nach 32 Bit gewandelt (**`int`**⇒**`long`**, **`word`**⇒**`dword`**)
- Enthält ein Ausdruck signed und unsigned gemischt wird die Operation nach der Expandierung unsigned durchgeführt.

```
unaryExpr ::=     monadicOperator castExpr | postfixExpr

monadicOperator   ::=
                  "+"  | "-" |        /* plus, minus */
                  "~"  | "!" |        /* bit not, logic not */
                  "++" | "--"                /* Increment, Decrement */

postFixExpr       ::=
                  primaryExpr                       |
                  postfixExpr "[" expr "]"   |
                  postfixExpr "++"                  |
                  postfixExpr "--"                  |
                  postfixExpr "." selection

primaryExpr       ::=
                  <varId>                           |
                  "THIS"                            |
                  <integer>                 |
                  <float>                           |
                  specialConst |
                  "(" expr ")" |
                  functionCall

selection ::=
                  intTypeSpecifier "(" <integer> ")" |
                  "CAN"                             |
                  "ID"                      |
                  "PGN"                     | (nur J1939)
                  "DLC"                             |
                  "TYPE"                            |
                  "DIR"                     |
                  "RTR"                     |
                  "TIME"                            |
                  "SIMULATED"         |
                  "PF"                      | (nur J1939)
                  "SA"                      | (nur J1939)
                  "DA"                      | (nur J1939)
                  "PRIO"                            | (nur J1939)
                  "R1"                      | (nur J1939)
                  "DP"                      | (nur J1939)
                  "CHARACTERISTIC"    | (nur J1939)
                  "ERRORCOUNTTX"      |
                  "ERRORCOUNTRX"      |
                  <signalSymbol> [ "." "PHYS" ] |
                  <attributeSymbol> |
                  <valueDescription> |

specialConstant ::=
                  "RX" | "TX" | "TXREQUEST" | "RXREMOTE" |        TXREMOTE |
                  "TXREQUESTREMOTE" | "RXDATA" | "TXDATA" | "TXREQUESTDATA" |
                  "NOW" | "LPT1" | "LPT2" | "LPT3" | "ERRORFRAME" | "PI"
```

Mit dem Schlüsselwort **this** wird innerhalb einer Event-Prozedur für den Empfang eines CAN-Objekts die Datenstruktur dieses Objektes bezeichnet.

Hinweis: **this** darf nicht verändert werden (read-only)! Um **this** aber als Parameter für CAPL-Prozeduren (siehe unten) verwenden zu können, wird dies nicht mehr generell verboten. Bei Schreibzugriffen auf **this** erscheint eine Warnung, daß die Änderung nur lokal erfolgt. Wird etwa nach einer Änderung der Befehl **output(this)** aufgerufen, leitet dies das unveränderte Original von **this** weiter!

Der Zugriff auf den Datenbereich einer Botschaft erfolgt über die Angabe das Datentyps als Selektor und den byte-wertigen Offset (bei "0" beginnend). Als Selektoren können **long**, **dword**, **int**, **word**, **char** und **byte** verwendet werden.

Bei Verwendung einer Projektdatenbasis kann auf die einzelnen Signale durch Angabe ihres Namens zugegriffen werden. Der Compiler übernimmt die Bestimmung des Bit-Offsets und der Bitlänge sowie die Umsetzung des Maschinenformats (Intel ⇔ Motorola). Die Werte werden im Rohformat interpretiert. Soll die physikalische Interpretation verwendet werden, so muß der Operator **.phys** angehängt werden. Der Compiler skaliert den Wert dann nach der in der Datenbasis angegebenen Umrechnungsformel (Faktor, Offset).

Beispiel:

```
message Motordaten m;
double x;
long i;
m.Drehzahl = 100;        // Zuweisung eines Rohwerts
m.Drehzahl.phys = 123.5; // mit phys. Umrechnungsformel
```

Hinweis: Beachten Sie, daß die Speicherung der Signalwerte immer diskret erfolgt! Wird ein physikalischer Wert zugewiesen, so wird nach der Skalierung der nächstliegende diskrete Rohwert eingetragen. Wird das Signal danach ausgelesen, so ist das Ergebnis nicht zwangsläufig der Ursprungswert.

Beispiel: Das Signal *Motordaten.Drehzahl* sei definiert als 16 Bit unsigned mit einem Offset von 0 und einem Faktor von 10. Wird als physikalischer Wert 1015 zugewiesen, müßte der Rohwert 1015/10 = 101.5 sein. Da nur diskrete Werte vorkommen, wird 101 gespeichert. Beim Auslesen wird dieser Wert zu 101*10 = 1010 skaliert.

Das Problem ist allgemeinerer Natur. Auch die Speicherung von Gleitkommazahlen in Variablen erfolgt bei Computern in einem diskreten Format (hier nach der IEEE-Spezifikation). Dadurch können Ungenauigkeiten entstehen:

```
double x, y;
x = 1.4;
y = 0.7;
if( x != y*2) write("Fehler x = %g, y*2 = %g",x,y*2);
```

In diesem Beispiel erfolgt u.U. eine Fehlermeldung. Der korrekte Vergleich lautet `if (abs(x-2*y) < 0.001) write ...`.

Für den Zugriff auf die Kontrollinformationen können folgende Selektoren verwendet werden:

Selektor	Bedeutung	mögliche Werte (Bemerkung)
ID	Botschafts-Identifier	
PG	Parameter-Group Number	(nur J1939)
CAN	Bausteinnummer	1, 2
DLC	Data-Length-Code	0, 1, ... ,8
DIR	Übertragungsrichtung, Event-Klassifikation	RX, TX, TXREQUEST.
RTR	Remote-Transmission-Request	0: kein RTR, 1: RTR
TYPE	Kombination aus DIR und RTR für eine effiziente Auswertung	(TYPE = (RTR << 8) | DIR)
TIME	Zeitpunkt, Einheit: 10 Mikrosekunden	
SIMULATED	Angabe ob Botschaft von einem simulierten Knoten gesendet wurde	0: nicht simulierter Knoten, 1: simulierter Knoten
PF		(nur J1939)
SA	Senderangabe im CAN-Identifier einer Parameter-Group	(nur J1939)
DA	Empfängerangabe im CAN-Identifier einer Parameter-Group	(nur J1939)
PRIO	Prioritätsangabe im CAN-Identifier einer Parameter-Group	(nur J1939 bei Initialisierung der Parameter-Group)
R1	R1 Feld im CAN-Identifier einer Parameter-Group	(nur J1939 bei Initialisierung der Parameter-Group)
DP	Data Page Feld im CAN-Identifier einer Parameter-Group	(nur J1939 bei Initialisierung der Parameter-Group)
CHARACTERISTIC	Zusammenfassung von DP, R1 und PRIO im CAN-Identifier einer Parameter-Group	(nur J1939: für Zugriffe außerhalb der Parameter-Group Initialisierung)
ERRORCOUNTTX	Sendefehlerzähler des CAN-Chips	
ERRORCOUNTRX	Empfangsfehlerzähler des CAN-Chips	

Um von der tatsächlichen Codierung von DIR und RTR unabhängig zu sein, stehen folgende symbolische Konstanten zur Verfügung:

Für DIR:

RX	Botschaft wurde empfangen	DIR == RX
TX	Botschaft wurde gesendet	DIR == TX
TXREQUEST	für Botschaft wurde Sendeanforderung gesetzt	(DIR == TXREQUEST)

Für TYPE:

RXREMOTE	Remote-Botschaft wurde empfangen	(DIR == RX) && RTR
TXREMOTE	Remote-Botschaft wurde gesendet	(DIR == TX)&&RTR
TXREQUESTREMOTE	für Remote-Botschaft wurde Sendeanforderung gesetzt	(DIR == TXREQUEST) && RTR
RXDATA:	Datenbotschaft wurde empfangen	(DIR == RX)&&!RTR
TXDATA:	Datenbotschaft wurde gesendet	(DIR == TX) &&!RTR
TXREQUESTDATA:	für Datenbotschaft wurde Sendeanforderung gesetzt	(DIR == TXREQUEST) && !RTR

In der Projektdatenbasis können den CAN-Objekten (Botschaften, Signale, …) Attribute und symbolische Wertebeschreibungen zugeordnet werden. Der Zugriff auf die Attributwerte der Objekte erfolgt mit dem Attributnamen als Selektor des Objekts.

Beispiel:

```
on message EngineSpeed {
   // 'msCycleTime' als Botschaftsattribut definiert
   if (this.msCycleTime == 100) { ... }

   // 'notAvailable' als Signalattribut definiert
   if (this.EngineSpeed.notAvailable) { ... }
}
```

Wertebeschreibungen können analog der Signalwerte verwendet werde. Damit ist es möglich symbolische Beschreibungen als Ersatz für Integer-Werte zu verwenden (analog einer *enum*-Definition in C).

Beispiel:

```
// Verwende Wertebeschreibung 'active' für Signal
                              'State'
message ABSDaten msg = { State = ABSDaten.State::active
                             };
```

In **on key *** Event-Prozeduren dient **this** zur Ermittlung des aktuellen Tastencodes.

Beispiel:

```
on key * {
  switch(this) {
    case ‘a’:   ...  break;
    case F10:  ... break;
    ...
  }
}
```

In Event-Prozeduren für die Veränderung des Fehlerzustands der CAN-Chips dient **this** der Ermittlung der Fehlerzähler des CAN-Chips.

Beispiel:

```
on errorPassive {
   write("ErrorCountTX = %d", this.errorCountTX);
}
```

Bei Event-Prozeduren, die auf Timer, Start oder Error-Frame reagieren, ist **this** nicht sinnvoll.

3.5.5 CAPL-Unterprogramme

In CAPL gibt es eine Reihe von eingebauten Funktionen, die sogenannten intrinsischem Funktionen (siehe unten). `caplFunction` bezeichnet vom Benutzer definierte Unterprogramme.

```
functionCall      ::=
                  intrinsicFunctionCall            |
                  caplFunctionCall                 |
                  externalFunctionCall

caplFunctionCall     ::=      <varId> "(" argExprList ")"
argExprList                   ::=    argExpr [ "," argExprList ]
argExpr                              ::=    expr    | <string>
```

Der Aufruf von selbst definierten Unterprogrammen erfolgt wie bei intrinsischen Funktionen.

```
caplProcedure     ::=
                  [resultType] <varId> "("[ funcParDeclList ]")" procBody

resultType ::=
                  intTypeSpecifier    |
                  "float"  | "double" | "void"

funcParDeclList ::=
                  funcParDecl [ "," funcParDeclList ] |
                  "void"

funcParDecl ::=
                  intTypeSpecifier declarator             |
                  "float"  | "double"                     |
                  "TIMER" <varId>                                 |
                  "MSTIMER" <varId>                       |
                  "MESSAGE" msgDenotation msgDeclarator |
                  "PG" msgDenotation msgDeclarator
```

procBody ist bei eventProcedures beschrieben. Einfache Typen (**int**, **float**, ...) werden als 'call by value' übergeben, strukturierte Typen (**message**, **pg**, **timer**, **array**) als 'call by reference'. Einfache Typen können per Funktionsergebnis (oder über globale Variable oder über ein Hilfskonstrukt **array** der Länge 1) zurückgegeben werden. Alle Variablen werden statisch angelegt. Funktionsparameter werden auf dem Stack übergeben. Dies ist bei rekursivem Aufruf von Funktionen zu berücksichtigen!

Wird der Ergebnistyp einer Funktion nicht angegeben, wird im Gegensatz zu C **void** angenommen. Die Parameterliste kann leer sein, dies entspricht dann einem **void**-Parameter.

Beispiele für Funktionsköpfe:

```
int func1(int i)            // Parameter und Ergebnis integer
int func2(int i, double x)  // Parameter integer und float,
                            // Ergebnis integer
void  func3(void)           // kein Parameter, kein Ergebnis
func4()                     // kein Parameter, kein Ergebnis
func5(char str[])           // Parameter char-array beliebiger
                            // Länge, kein Ergebnis
float func6(int matrix[][]) // Parameter 2-dim. Matrix,
                            // Ergebnis float
```

Funktionen können mit Overloading definiert werden. Das bedeutet, daß mehrere Funktionen mit jeweils gleichem Namen aber unterschiedlichen Parametern deklariert werden können: Der Ergebnistyp muß jeweils gleich sein.

```
void print(int i)
void print(int j[])
void print(int j[][]);
```

Auf diese Art kann z.B. eine einheitliche Ausgabefunktion für alle Datentypen definiert werden, die immer unter gleichem Namen aufgerufen wird. Der Compiler ruft dann immer automatisch die zum Datentyp passende Funktion auf. Auch die vorhandenen intrinsischen Funktionen lassen sich so erweitern (z. B. **`write(int)`**, **`write(float)`**,...,**`output(float)`**,...).

Verwendung von Arrays als Parameter:

Arrays werden formal wie in C übergeben. Als Erweiterung ist aber auch die Übergabe mehrdimensionaler Arrays möglich. Die Anzahl der Elemente einer Dimension läßt sich mit der Funktion **`elCount()`** erfragen. Der Compiler führt einen Bereichsüberprüfung durch. Wird ein Index kleiner null oder größer als die Zahl der Elemente verwendet, wird ein Laufzeitfehler über die Funktion **`runError()`** erzeugt.

Beispiel:

```
void printMatrix(int m[][]) {
   int i,j;
   for(i = 0; i < elCount(m); i++) {
      for(j = 0; j < elCount(m[0]); j++) {
         write("%ld", m[i][j]);
      }
   }
}
```

3.6 Spezielle Themen in CAPL

3.6.1 Einbinden einer Windows DLL in CAPL

Sie können Funktionen, die Sie in einer eigenen Windows DLL implementiert haben, in CAPL-Programmen aufrufen. Die Funktion aus der DLL werden dabei über eine Funktionstabelle exportiert.

Für das Einbinden einer DLL in der 16-bit-Version beachten Sie bitte folgenden Hinweis: Die Funktionen dieser DLL, die aus der Echtzeitbibliothek (bei CANoe im Simulationsaufbau, bei CANalyzer im Sendezweig) aufgerufen werden, werden in einer Interrupt Service Routine durchlaufen. Dies bedeutet insbesondere:

- Aller Speicher, auf den zugegriffen wird, muß FIXED sein. Dafür ist im Projekt das modifizierte Objekt der Borland-RTL `WINMEM.OBJ` einzubinden. (vgl. die Moduldefinitionsdatei.)
- Dateizugriffe und Aufrufe des Windows-API sind nicht erlaubt!

Diese Einschränkungen gelten nicht für die 32-bit Version des Programms.

Für den Compiler und den Browser muß der Name der DLL in der Datei `CAN.INI` im Abschnitt `[SIMULATION]` die DLL angegeben werden. Heißt Ihre DLL beispielsweise `CAPLDLL.DLL`, so fügen Sie in die Datei `CAN.INI` folgenden Abschnitt ein:

```
[SIMULATION]
CAPLDLL1=capldll.dll
```

Hinweis: Mit dem Einbinden eigener DLLs in CAPL-Programme stehen Ihnen alle Wege offen, auf Systemressourcen (Speicherverwaltung, Hardwarezugriffe etc.) zuzugreifen. Wegen der Vielfalt damit verbundener Probleme, sollten jedoch nur erfahrene Programmierer von diesem Möglichkeiten Gebrauch machen. Bitte haben Sie Verständnis dafür, daß Vector für das Erstellen eigener DLLs keinen Support leisten kann.

Beschreibung der CAPL-Exportabelle

Name:

CAPL_DLL_INFO_LIST

Funktion:

Export von Funktionen nach CAPL

Beschreibung:

Spalte 1 enthält den Namen der zu exportierenden Funktion

Spalte 2 enthält die Adresse der Funktion in der DLL. Alle Funktionen müssen mit PASCAL-Calling-Konvention definiert und exportiert werden.

Spalte 3 enthält eine Beschreibung der Funktionsparameter und Return-Werte. Folgende Zeichen sind möglich:

Zeichen	Bedeutung	Bemerkung
V	void	
Ll	long	
D	unsigned long	
I	int	nur zusammen mit Feldgröße > 0
W	unsigned int	nur zusammen mit Feldgröße > 0
B	unsigned char	nur zusammen mit Feldgröße > 0
C	char	nur zusammen mit Feldgröße > 0
F	float	bedeutet „double“ im 8 Byte 80387 Format)
0xabcd	Zeichen für gültige Tabelle	

Ein Zeiger auf die Adresse dieser Tabelle muß in der Funktion `caplDllGetTable()` zurückgegeben werden. Diese Funktion muß mit extern C-Linkage übersetzt werden.

Hinweis: Rückgabewerte (`long`, `int`) der Funktionen müssen in das Register _EAX kopiert werden. Beachten Sie, daß außer den Feldern alle Parameter vom Typ `long` sind.

Beispiel:

```
unsigned long _export  pascal caplDllGet(void)
{
     unsigned long temp;
     calcRc(&temp);
     _EAX = temp;
     return _EAX;
}
```

Das vollständige Beispiel finden Sie im Unterverzeichnis CAPLDLL Ihres Systemverzeichnisses.

3.6.2 Behandlung von Laufzeitfehlern

Eine Reihe von Laufzeitfehlern wird in CAPL überwacht:

- Division durch Null
- Über- oder Unterschreiten von Array-Grenzen
- Über- oder Unterschreiten von Offsets in den Datenbereich von Botschaften
- Stackoverflow beim Aufruf von CAPL-Unterprogrammen

Wird ein Laufzeitfehler entdeckt, wird die intrinsische Funktion **runError()** aufgerufen. Diese gibt einen Hinweis im Write-Fenster aus, der den Namen des CAPL-Programms, die Fehlerart und einen Fehlerindex enthält. Mit Hilfe des Fehlerindex wird die Stelle im CAPL-Quelltext gefunden, die den Fehler verursachte. Nach Ausgabe der Meldung wird die Messung abgebrochen.

Die Funktion **runError()** kann auch vom Benutzer direkt aufgerufen werden, um Assertions zu generieren.

3.7 Liste der eingebauten Funktionen

Es folgt eine nac Themengebieten geordnete Liste der vordefinierten Funktionen („Intrinsic Functions“).

3.7.1 Zugriff auf CAN und andere Hardware

output(message msg)
output(errorFrame)

Funktion:

gibt eine Botschaft oder einen Error-Frame aus dem Programmblock aus.

Parameter:

Variable vom Typ "message" oder "errorFrame"

Beispiel:

```
message 0x100 m100;
...
output (m100);
output (errorFrame);
```

resetCan()

Funktion:

Reset der CAN-Controller. Kann zum Rücksetzen der CAN-Controller nach einem BUSOFF oder zur Aktivierung von Konfigurationsänderungen verwendet werden. Da die Ausführung der Funktion einige Zeit benötigt und der CAN-Controller kurzfristig vom Bus abgekoppelt wird, können hierbei Botschaften verlorengehen.

Beispiel:

```
on key 'r' {
   // nach BUSOFF wird Controller zurückgesetzt
   resetCan();
}
```

setBtr(long channel, byte btr0, byte btr1)

Funktion:

Setzen einer anderen Baudrate. Die Werte werden erst beim nächsten Aufruf der Funktion "resetCan()" aktiv. Die Funktion kann auch zum automatischen Ermitteln der Baudrate verwendet werden.

Es ist zu beachten, daß die Werte vom verwendeten CAN Controller abhängen.

Parameter:

CAN-Kanal:

0: beide Controller

1: Kanal 1

2: Kanal 2

BTR0: Wert des Bit Timing Register 0

BTR1: Wert des Bit Timing Register 1

Beispiel:

```
setBtr(0, 0x00, 0x3a);  // 500 kBaud bei 82C200
resetCan();             // aktivieren
```

setOcr(long channel, byte ocr)

Funktion:

Setzen des Output Control Registers. Die Werte werden erst beim nächsten Aufruf der Funktion "resetCan()" aktiv.

Es ist zu beachten, daß die Werte von der verwendeten CAN Plattform abhängen.

Parameter:

CAN-Kanal:

0: beide Controller

1: Kanal 1

2: Kanal 2

OCR: Wert des Output Control Registers

Beispiel:

```
setOcr(0, 0x02);              // setzen
resetCan();                   // aktivieren
```

Setzt bei Verwendung der CAN AC2/200/ANA die CAN Controller auf „passive monitoring“.

outport (word addr, byte value)

Funktion:

Ausgabe eines Bytes an einen Port.

Parameter:

Port-Adresse bzw. "LPT1"/"LPT2"/"LPT3" und auszugebender Wert.

Beispiel:

```
outport (0x3f8, 12); /* gibt 12 an Port 0x3f8 aus */
outport (LPT2, 'x'); /* gibt 'x' an LPT2 aus */
```

byte inport(word addr)

Funktion:

Einlesen eines Bytes von einem Port.

Parameter:

Port-Adresse

Rückgabe:

eingelesenes Byte

Beispiel:

```
val = inport (0x3f8); /* liest Port 0x3f8 aus */
```

Hinweis: Die CAPL-Funktionen **`inport()`** und **`outport()`** ermöglichen auch für die 32-bit Version unter Windows 95 und Windows NT den Zugriff auf die parallele Schnittstelle. Unter Windows NT muß dazu zunächst ein Treiber für den Parallelport installiert werden, den Sie zusammen mit den Installationshinweisen im Verzeichnis Exec32\GpIoDrv finden.

long getCardType()

Funktion:

Ermittelt den Typ der verwendeten CAN-Plattform. Wird zum Beispiel zur Programmierung der BTR / OCR Werte benötigt.

Rückgabe:

Wert	Board-Typ
0	DBB196 - Daimler-Benz-Board FullCAN
1	DBB196B - " mit BasicCAN
2	CANIB - Bosch CANIB
3	DEMO - Demo-Treiber
6	CANAC2 - Softing AC2/200/ANA
7	CANAC2X - Softing AC 2/527/ANA
8	CPC/PP = EMS Wünsche Modul
9	INDIGO - Silicon Graphics Indigo2
10	CANCARD - PCMCIA 11 Bit
11	CANCARDX - PCMCIA 29 Bit
12	CANAC2B - Softing AC2/527 11 Bit
13	VAN462 - NSI VAN Card
14	VANDEMO - VAN Demo Treiber
16	Vector CAN-Dongle
17	Vector CANCardx
18	Virtueller CAN-Treiber
19	Caesar
20	Softing CANcard-SJA

Es können weitere Typen hinzukommen!

Beispiel:

```
switch( getCardType( ) ) {
   case 6:  setOcr(0,0x02);break;
   case .....
   default:
      write("Unbekannter Treiber %d",getCardType());
      break;
}
```

long getChipType(long channel)

Funktion:

Ermittelt den Typ des verwendeten CAN-Controllers.

Parameter:

Wert:	CAN-Kanal:
0:	beide Controller
1:	Kanal 1
2:	Kanal 2

Rückgabe:

Wert	Typ des Controllers
5	NEC 72005
200	Philipps PCA82C200
462	MHS29C462 VAN Controller
526	Intel 82526
527	Intel 82527
1000	SJA 1000 (CANcardX)
1001	SJA 1000 (CANcard)

Es können weiter Typen auftreten. Demo-Versionen liefern das Ergebnis 0 oder simulieren einen der existierenden Typen. Wird auf einen nicht existierenden Kanal zugegriffen (z.B. Kanal 2 bei CPC/PP) oder unterstützt der verwendete Treiber diese Funktion nicht, ist das Funktionsergebnis 0.

3.7.2 Benutzerinteraktion

write (char format[], ...)

Funktion:

gibt eine textuelle Meldung in das Write-Fenster aus.

Parameter:

Format-String, Variablen oder Ausdrücke

Write ist an die C-Funktion "printf" angelehnt.

Erlaubte Formatangaben:

"%ld","%d" :	Dezimaldarstellung
"%lx","%x" :	Hexadezimaldarstellung
"%lX","%X":	Hexadezimaldarstellung in Großbuchstaben
"%lu","%u" :	vorzeichenlose Dezimaldarstellung
"%lo","%o" :	Oktaldarstellung
"%s" :	String-Ausgabe
"%g","%lf" :	Floatingpoint
"%c" :	Ausgabe eines Characters
"%%" :	Ausgabe von %

Der Compiler kann den Format-String nicht überprüfen. Unerlaubte Formatangaben führen zu undefinierten Ergebnissen.

Beispiel:

```
i = 10; j = 12;
write ("d = %ld, h = 0%lxh",i,j);
/* ergibt: „d = 10, h = 0ch“ */
```

beep(int freq, int duration);

Funktion

Tonausgabe. In der Windows CANalyzer-Version hat der Parameter *duration* keine Auswirkung.

Parameter

freq = Integer für Tonhöhe.

In der Windows Version bestimmt der Parameter *freq*, welcher Ton ausgegeben wird. In der Datei WIN.INI sind die Töne im Abschnitt [SOUND] definiert. Der Eintrag wird in Klammern angegeben.

Wert für freq	Bedeutung
0x0000	SystemDefault
0x0010	SystemHand
0x0020	SystemQuestion
0x0030	SystemExclamation
0x0040	SystemAsterisk
0xFFFF	Standard Beep

Rückgabe: keine

Hinweis: Wenn in Ihrem Computer keine Soundkarte installiert ist, erzeugt Windows bei Aufruf der Funktion immer den gleichen Signalton. Der Parameter *freq* hat in diesem Fall also keinen Effekt.

sysExit()

Funktion:

Beendet das System (CANalyzer/CANoe) aus einem CAPL-Programm heraus.

sysMinimize()

Funktion:

Minimiert bzw. restauriert das Anwendungsfenster des CANalyzers bzw. von CANoe. Beim ersten Aufruf der Funktion nach Systemstart wird das Anwendungsfenster minimiert, danach wird es abwechselnd auf Normalgröße wiederhergestellt und minimiert.

dword keypressed()

Funktion:

Diese Funktion liefert den Tastencode einer aktuell gedrückten Taste. Ist keine Taste gedrückt, liefert sie 0. Hiermit kann zum Beispiel in einer Timer-Funktion das Drücken einer Taste erfragt werden. Es kann auch auf das Loslassen einer Taste reagiert werden.

```
variables {
   msTimer Zeitgeber;         // Timer
   message 100 botschaft;     // CAN-Botschaft
}

on key F1 {
   setTimer(Zeitgeber,100);   // 100ms Timer starten
   write("F1 gedrückt");      // Ausgabe Write-Fenster
}

on timer Zeitgeber {
   if(keypressed()) {
      setTimer(Zeitgeber,100); // Timer neu starten
      // Senden solange Taste gedrückt
      output(botschaft);
   }
   else {
      // Reaktion auf Loslassen der Taste
      write("F1 losgelassen");
   }
}
```

3.7.3 Logging-Funktionen

trigger()

Funktion:

Auslösen der Logging-Triggerung.

Beispiel:

```
on key 't' // Triggern auf Tastendruck
{
   trigger();
}
```

Hinweis: Beachten Sie, daß der Aufruf der Funktion **`trigger()`** aus einem CAPL-Programm im Logging-Zweig oder aus dem Simulationaufbau von CANoe bzw. dem Meßaufbau des CANalyzers heraus erfolgen muß.

void writeToLog(char format[], ...);
void writeToLogEx(char format[], ...);

Funktion:

Gibt eine textuelle Meldung in eine ASCII-Logging-Datei aus. Die Funktionen sind an die C-Funktion `printf()` angelehnt. Der Compiler kann den Format-String nicht überprüfen. Unerlaubte Formatangaben führen zu undefinierten Ergebnissen.

Die Funktion **writeToLog()** gibt am Beginn der Zeile ein Kommentarzeichen (//) und eine Zeitangabe aus. Die Funktion **writeToLogEx()** überläßt das Ausgabeformat dagegen vollständig dem Anwender.

Parameter:

Format-String, Variablen oder Ausdrücke

Erlaubte Formatangaben: (vgl. Funktion **write()**)

setLogFileName(char filename[])

Funktion:

Setzt den Namen der Logging-Datei

Parameter:

Neuer Name der Logging-Datei Der Name kann als absoluter Pfad oder als reiner Dateiname angegeben werden. Wird ein Pfad angegeben, werden alle Order des Pfads angelegt, soweit sie noch nicht existieren. Bei Angabe eines Dateinamens, wird die Logging-Datei im Verzeichnis der aktuellen Konfiguration angelegt. Die Verzeichnisse innerhalb des Pfads müssen mit einem doppelten Backslash ('\\') getrennt werden. Der Dateiname darf keine Dateierweiterung enthalten, diese automatisch vom System ermittelt.

Beispiel:

```
// Setzt den Namen der Logging-Datei auf "newlog"
// im Verzeichnis der aktuellen Konfiguration
setLogFileName("newlog");
...
// Setzt den absoluten Pfad der Logging-Datei
setLogFileName("c:\\canw\\demo\\automot\\newlog");
...
```

3.7.4 Standard / Extended Identifier

long isStdId(dword id)
long isStdId(message m)
long isExtId(dword id)
long isExtId(message m)

Funktion:

Prüft den Parameter auf Extended-Identifier (29 Bit) bzw. Standard-Identifier (11 Bit)

Parameter:

Variable vom Typ "message" oder Id-Teil einer Message

Rückgabe:

1 falls Prüfung erfolgreich, sonst 0

Beispiel:

```
if(isExtId(this)) { ... }
std = isStdId(m100.id);
```

long valOfId(dword id)
long valOfId(message* m)

Funktion:

Liefert den Wert eines Botschafts-Identifiers unabhängig von seinem Typ.

Parameter:

Variable vom Typ **`message*`** oder Id-Teil einer Message

Rückgabe:

Identifier als Long-Größe

Beispiel:

```
id = valOfId(this); // funktioniert auch mit ext.Ids
```

dword mkExtId(dword id)

Funktion:

Liefert einen extended Identifier

Parameter:

Id-Teil einer Message

Rückgabe:

Extended Identifier

Beispiel:

```
msg.id = mkExtId(this.id);
```

3.7.5 Ablaufkontrolle

stop()

Funktion:

Im Online-Modus wird die laufende Messung gestoppt. Im Offline-Modus bewikt der Aufruf von **`stop()`** eine programmierte Unterbrechung der laufenden

Messung. Diese wird danach durch die Hauptmenüfunktionen *Start|Start*, *Start|Animate*, oder *Start|Step* wiederaufgenommen.

Beispiel:

```
stop();
```

canOnline()
canOffline()

Funktion:

Die Funktion **canOffline()** koppelt den Knoten vom Bus ab; das Senden und Empfangen von Botschaften ist danach nicht mehr möglich. Nach dem Aufruf der Funktion kann der Knoten mit der Funktion **canOnline()** wieder mit dem Bus verbunden werden. Vom Knoten gesendete Botschaften werden dann wieder auf den Bus weitergeleitet.

Hinweis: Diese Funktionen sind nur im Simulationsaufbau von CANoe verfügbar. Sie können dort beispielsweise eingesetzt werden, um in Restbussimulationen mit Netzmanagement sicherzustellen, daß ein in CAPL simulierter Knoten nach dem Übergang in die Busruhe keine Botschaften mehr sendet.

3.7.6 Zeitverwaltung

setTimer(msTimer t, long duration)
setTimer(timer t, long duration)

Funktion:

Setzen eines Timers

Parameter:

Timer- bzw. msTimer-Variable und ein Ausdruck, der die Dauer des Timer spezifiziert.

Beispiel:

```
/* msTimer t;  Timer t1; */
setTimer(t, 200); /* Den Timer t auf 200 ms setzen */
setTimer (t1, 2);  /* Den Timer t1 auf 2 s setzen */
```

cancelTimer(msTimer t)
cancelTimer(timer t)

Funktion:

Stoppen eines laufenden Timers

Parameter:

Timer- bzw. msTimer-Variable

Beispiel:

```
/* msTimer t;  Timer t1; */
cancelTimer(t); cancelTimer(t1);
```

Hinweis: Ein Aufruf von **`cancelTimer()`** ist verhältnismäßig zeitaufwendig. Die Funktion sollte daher sparsam eingesetzt werden.

setMsgTime(message m, NOW)
setMsgTime(message m1, message m2)

Funktion:

ordnet einer Botschaft die Erfassungszeit einer anderen oder die aktuelle Zeit zu. Die Funktion ist obsolet und dient nur noch der Kompatibilität zu älteren Versionen.

Parameter:

Variable vom Typ "message" oder "now".

Beispiel:

```
setMsgTime(m100, this); /* CANalyzer 1.xx & 2.xx*/
m100.time = this.time;  /* CANalyzer 2.xx */
setMsgTime (m101, now);
```

dword timeNow()

Funktion:

Liefert die aktuelle Systemzeit

Rückgabe:

Systemzeit in Einheiten von 10 µsec. Diese Zeit wird mit Hilfe des PC-Timers mit einer Auflösung von 1 msec bestimmt.

Beispiel:

```
float x;
x = timeNow()/100000.0; //aktuelle Zeit in Sekunden
```

long timeDiff(message m1, NOW)
long timeDiff(message m1, message m2)

Funktion:

Differenzzeit zwischen Botschaften oder einer Botschaft und der aktuellen Zeit in ms. Die Differenz kann auch direkt in der Einheit 10 Mikrosekunden berechnet werden (**`msg2.time-msg1.time`** bzw. **`now-msg1.time`**).

Parameter:

1. Variable vom Typ **`message`**
2. Variable vom Typ **`message`** oder **`now`**.

Rückgabe:

Zeitdifferenz in ms;

Beispiel:

```
diff = timeDiff(m100,now);  // Zeitdiff. mit timeDiff()
diff = this.time-m100.time; // oder aber direkt
```

getLocalTime(long time[])

Funktion:

Liefert Angaben über die aktuelle Zeit und das aktuelle Datum in einem Array vom Typ long zurück.

Die Komponenten des Arrays werden mit folgenden Informationen gefüllt:

Nr.	Bedeutung	Wertebereich
1.	Sekunden	0 - 60
2.	Minuten	0 - 60
3.	Stunden	0 - 24
4.	Tag im Monat	1 - 31
5.	Monat	0 - 11
6.	Jahr	beginnend bei 1900
7.	Wochentag	0 - 7
8.	Tag im Jahr	0 - 365
9.	Flag für Sommerzeitangabe	0 - 1, 1 = Sommerzeit

Parameter:

Ein Array vom Typ long mit mindestens 9 Einträgen.

Beispiel:

```
long tm[9];
getLocalTime(tm);
// tm enthält jetzt folgende Einträge:
// tm[0] = 3;  (Sekunden)
// tm[1] = 51; (Minuten)
// tm[2] = 16; (Stunden)
// tm[3] = 21; (Tag im Monat)
// tm[4] = 7;  (Monat beginnent mit 0)
// tm[5] = 98; (Jahr)
// tm[6] = 5;  (Wochentag)
// tm[7] = 232;(Tag im Jahr)
// tm[8] = 1;  (Sommerzeit)
```

getLocalTimeString(char timeBuffer[]);()

Funktion:

Liefert das aktuelle Datum und die aktuelle Uhrzeit in der Form *ddd mmm dd hh:mm:ss jjjj* (z.B. "Fri Aug 21 15:22:24 1998“) im zur Verfügung gestellten Puffer zurück.

Parameter:

Puffer in den die Zeit geschrieben wird. Der Puffer muß mindestens 26 Zeichen groß sein.

Beispiel:

```
char timeBuffer[64];
getLocalTimeString(timeBuffer);
// in timeBuffer steht jetzt z.B.
// "Fri Aug 21 15:22:24 1998"
```

3.7.7 Zugriff auf CANoe-Umgebungsvariablen

int getValue(EnvVarName)
float getValue(EnvVarName)
int getValue(EnvVarName, char buffer[])

Funktion:

Ermittelt den Wert der Umgebungsvariable mit dem Bezeichner *EnvVarName*. Der Typ des Rückgabewerts richtet sich dabei nach dem Typ der Umgebungsvariable (**`int`** für diskrete (Form 1) und **`float`** für kontinuierliche Umgebungsvariablen (Form 2)). Bei Zeichenketten-Umgebungsvariablen (Form 3) wird der aktuelle Wert in einen Puffer geschrieben, den Sie beim Funktionsaufruf mitgeben.

Parameter:

Name der Umgebungsvariable und für Form 3 Rückgabepuffer und Größe des Puffers.

Rückgabe:

aktueller Wert der Umgebungsvariable (für Form 1 und Form 2)

Beispiel:

```
int val;
float fval;
char buff[25];

// val den Wert der Umgebungsvar. 'Schalter' zuweisen
val = getValue(Schalter);
// fval den Wert der Umgebungsvar. 'Temperatur' zuweisen
val = getValue(Temperatur);
// Wert der Umgebungsvariablen 'KnotenName' lesen
getValue(KnotenName,buff);
```

putValue(EnvVarName,int val)
putValue(EnvVarName,float val)
putValue(EnvVarName,char val[])

Funktion:

Weist der Umgebungsvariablen mit dem Bezeichner *EnvVarName* den Wert *val* zu. Diskreten Umgebungsvariablen werden Integers, kontinuierlichen Umgebungsvariablen werden Gleitkommazahlen und Zeichenketten-Umgebungsvariablen werden Zeichenketten zugewiesen.

Parameter:

Umgebungsvariablenname neuer Wert der Umgebungsvariable

Beispiel:

```
// Umgebungsvariablen ‘Schalter’ den Wert 0 zuweisen
putValue(Schalter,0);
// Umgebungsvar. ‘Temperatur’ den Wert 22.5 zuweisen
putValue(Temperatur,22.5);
// Umgebungsvar. ‘KnotenName’ den Wert "Master" zuweisen
putValue(KnotenName, "Master");
```

callAllOnEnvVar()

Funktion:

Führt die Ereignisprozeduren **on envVar ...** für alle Umgebungsvariablen aus. Dies kann beim Messungsstart notwendig sein, um Umgebungsvariablen zu initialisieren, Timer aufzusetzen, die bei Änderungen von Umgebungsvariablenwerten aktiviert werden, oder Botschaften mit Startwerten von Umgebungsvariablen auf den Bus zu legen.

Beispiel:

```
on start {
   write("Ampelmodul gestartet");

   // call all envvar procedures of this model and
   // thus consider the START VALUES of all environment
   // variables for:
   //     initialization of all message variables
   //     starting of any timers
   //     sending messages (output) with start values
   CallAllOnEnvVar();
}
```

3.7.8 Intel/Motorola-Formate

Die Arithmetik in CAPL erfolgt im Little-Endian-Format (Intel). Die folgenden Swap-Funktionen dienen zur Vertauschung von Bytes beim Übergang von und zum Big-Endian-Format (Motorola):

word swapWord(word x)
int swapInt(int x)
dword swapDWord(dword x)
long swapLong(long x)

Funktion:

Vertauschung der Bytes des Parameters

Parameter:

Wert, dessen Bytes zu vertauschen sind.

Rückgabe:

Wert mit vertauschten Bytes.

Beispiel:

```
bigEndian = swapInt(1234); /* Erzeugung der Konstanten
                              1234 für Motorola-Rechner */
```

3.7.9 Trigonometrie und mathematische Funktionen

double sin(double x)

Funktion:

Berechnung des Sinus.

Parameter:

Wert in Radiant, dessen Sinus zu berechnen ist.

Beispiel:

```
x = sin(PI);                  /* Ergebnis 0.0 */
```

double cos(double x)

Funktion:

Berechnung des Cosinus.

Parameter:

Wert in Radiant, dessen Cosinus zu berechnen ist.

Beispiel:

```
x = cos(PI);                  /* Ergebnis -1.0 */
```

double sqrt(double x)

Funktion:

Berechnung der Quadratwurzel

Parameter:

Wert, dessen Quadratwurzel zu berechnen ist.

Beispiel:

```
x = sqrt(4.0);                /* Ergebnis: 2.0*/
```

double exp(double x)

Funktion:

Berechnung der Exponentialfunktion

Parameter:

Exponent zur Basis e

Beispiel:

```
x = exp(1.0);                    /* Ergebnis: 2.7182... */
```

long abs(long x)
double abs(double x)

Funktion:

Liefert den Absolutbetrag

Parameter:

Wert, dessen Absolutbetrag zurückgeliefert wird.

Rückgabe:

Absolutbetrag von x.

dword random(dword x)

Funktion:

Berechnung einer Zufallszahl zwischen 0 und x

Parameter:

Maximaler Wert, den die Funktion liefert

Beispiel:

```
x = random(2500); // erzeugt Zufallszahl zw. 0 und 2500
```

3.7.10 Dateifunktionen

Es besteht häufig der Bedarf, Variablen oder Meßgrößen über mehrere Messungen hinweg zu speichern. Hierzu wurden Dateifunktionen für CAPL verfügbar gemacht. Die verwendeten Dateien haben die gleiche Struktur wie das weitverbreitete Format der INI-Dateien von MS-Windows. Die Dateizugriffe wurden auf einfache Anwendbarkeit und Sicherheit konzipiert. Sie sind daher nicht sehr schnell.

Ihre Anwendung bietet sich in der *on start*-Event-Prozedur bzw. bei Änderung eines Wertes an.

Um Echtzeitfähigkeit zu gewährleisten müssen die verwendeten Dateien vor Start der Messung geladen werden. Dazu dient die Datei `CAPL.INI` im CANalyzer-Systemverzeichnis (z.B. `C:\CANWIN\EXEC`). In ihr muß jede Datei angegeben werden, auf die während der Messung zugegriffen werden soll. Vorsicht: Greifen Sie nicht auf die Dateien `WIN.INI` und `SYSTEM.INI` im Windows-Systemverzeichnis zu.

Beispiel einer solchen Datei `CAPL.INI`:

```
[CAPL-INI]
Display=0   // Display=1: Display files that will be loaded
            // in Write-Window after start of measurement
MaskCount=2 // number of mask entries in this file

// directory where CAPL data files will be searched
FilePath=d:\usr\CANwin\exec

Mask1=*.dat // files that will be accessed from CAPL
            // (all files with extension *.dat will be loaded)
// file that will be accessed from CAPL
Mask2=capl2.cdt // (file capl2.cdt will be loaded)
```

long fileWriteString(char section[], char entry[], char value[], char filename[])

Funktion:

Öffnet die Datei *filename*, sucht die Sektion *section* und schreibt die Variable *entry* mit dem Wert *value*. Existiert *entry* schon, wird der alte Wert überschrieben. Funktionsergebnis ist die Zahl der geschriebenen Zeichen bzw. 0 bei Fehler.

long fileWriteInt(char section[], char entry[], long def, char filename)

Funktion:

Analog zur Funktion **`fileWriteString()`**, schreibt aber statt eines Textes eine Long-Variable in die Datei. Ist das Funktionsergebnis 0, trat ein Fehler auf, sonst Erfolg.

Beispiel:

```
fileWriteInt("DeviceData","DeviceAddr",2,  "TEST.INI");
```

Dieser Aufruf schreibt folgenden Eintrag in die Datei `TEST.INI`:

```
[DeviceData]
DeviceAddr=2
```

long fileWriteFloat(char section[], char entry[], float def, char filename[])

Funktion:

Analog zur Funktion **`fileWriteString()`**, schreibt aber statt eines Textes eine Float-Variable in die Datei.

long fileReadString(char section[], char entry[], char def[], char buffer[], long bufferlen, char filename[])

Funktion:

Sucht in der Datei *filename* in der Sektion *section* die Variable *entry*. Ihr Inhalt (Wert) wird in den Puffer *buffer* geschrieben. Dessen Länge *muß* in *bufferlen* korrekt übergeben werden. Wird die Datei oder *entry* nicht gefunden, wird der Vorgabewert *def* in *buffer* kopiert.

long fileReadInt(char section[], char entry[], long def, char filename[])

Funktion:

Sucht in der Datei *file* in der Sektion *section* die Variable *entry*. Ist ihr Wert eine Zahl, wird diese Zahl als Funktionsergebnis geliefert. Wird die Datei oder *entry* nicht gefunden oder enthält *entry* keine gültige Zahl, wird der Vorgabewert *def* als Funktionsergebnis geliefert.

Beispiel:

```
myAddress=fileReadInt(
          "DeviceData","DeviceAddr",0,"TEST.INI");
```

Erfolgt dieser Aufruf nach der Schreibanweisung im Beispiel zu **`fileWriteInt()`**, wird der Variablen *myAddr* der Wert 2 zugewiesen. Existiert der Eintrag *DeviceAddr* nicht in der Datei `TEST.INI`, wird der Vorgabewert 0 zugewiesen.

float fileReadFloat(char section[], char entry[], float def, char filename[])

Funktion:

Analog zur Funktion **`fileReadInt()`** für Gleitkommazahlen.

long fileReadArray(char section[], char entry[], char buffer[], long bufferlen, char filename[])

Funktion:

Sucht in der Datei *file* in der Sektion *section* die Variable *entry*. Deren Inhalt wird als Liste von Bytewerten interpretiert. Das Zahlenformat ist dezimal oder bei vorangestelltem 0x hexadezimal. Die Zahlen sind durch Leerzeichen, Tabulatoren, Komma, Strichpunkt oder Schrägstrich getrennt. Der *buffer* wird mit bis zu *bufferlen* Bytes gefüllt.

Rückgabe:

Anzahl der geschriebenen Zeichen.

Beispiel:

Datei `TEST.INI`:

```
[DATA]
FIELD=1,2,3,0x20,100
```

```
int len;
char buf[20];
len = fileReadArray(
           "DATA","FIELD",buf,elCount(buf),"TEST.INI");
```

Ergebnis *len* ist 5. Der Array *buf* wird mit den Werten 1, 2, 3, 32, 100 gefüllt.

3.7.11 Sequentieller Dateizugriff

Der CAPL-Compiler bietet Ihnen verschiedene Funktionen zum sequentiellen Zugriff auf Dateien:

long seqFileLoad(char fileName)

Funktion:

Die Funktion öffnet die Datei mit Namen *fileName* für den Lesezugriff.

Rückgabe:

Der Rückgabewert ist der Datei-Handle, der bei den Leseoperationen als Parameter übergeben werden muß. Falls ein Fehler auftritt, ist der Rückgabewert ein Fehlercode <= 0.

long seqFileClose(long file)

Funktion:

Die Funktion schließt die durch den Handle *file* angegebenen Datei. Alle vom System belegten Puffer werden beim Schließen freigegeben.

Rückgabe:

Bei Erfolg gibt die Funktion den Wert 0 zurück. Beim Auftreten eines Fehlers wird ein von 0 verschiedener Fehlercode zurückgegeben.

long seqFileGetBlock(char buffer[], dword bufferSize, long file)

Funktion:

Die Funktion liest höchstens *bufferSize* Zeichen von der durch den Handle *file* angegebenen Datei in das Feld *buffer*. Der Positionszeiger wird dabei um die Anzahl der erfolgreich gelesenen Zeichen inkrementiert. Nach einem Fehler ist der Wert des Positionszeigers unbestimmt.

Rückgabe:

Der Rückgabewert enthält die Anzahl der erfolgreich gelesenen Zeichen. Er kann kleiner als *bufferSize* sein, falls ein Fehler auftritt oder das Dateiende erreicht wird.

long seqFileGetLine(char buffer[], dword bufferSize, long file)
long seqFileGetLineSZ(char buffer[], dword bufferSize, long file)

Funktion:

Die Funktion liest höchstens *bufferSize*-1 Zeichen von durch den Handle *file* angegebenen Datei in das Feld *buffer*. Nach einem Zeilenende- oder Dateiende-Zeichen werden keine weiteren Zeichen mehr gelesen. Das Zeilenende-Zeichen wird nicht in *buffer* kopiert.

Die Funktion **`seqFileGetLine()`** schließt die Zeichenkette wird *nicht* mit '\0' ab; **`seqFileGetLineSZ()`** schließt die Zeichenkette mit '\0' ab.

Rückgabe:

Der Rückgabewert enthält die Anzahl der erfolgreich gelesenen Zeichen. Er ist negativ, falls ein Fehler auftritt.

Hinweis: Die CAPL-Funktion **`seqFileGetLineSZ()`** entspricht der Funktion `fgets()` aus der Standard C-Bibliothek.

long seqFileRewind(long file)

Funktion:

Die Funktion setzt die durch den Handle *file* angegebenen Datei an den Anfang zurück.

Rückgabe:

Bei Erfolg gibt die Funktion den Wert Null zurück. Beim Auftreten eines Fehlers wird ein von Null verschiedener Fehlercode zurückgegeben.

Beispiel

Die Testdatei `TEST.DAT` wird in dem CAPL-Beispielprogramm beim Messungsstart geöffnet und nach Drücken der Taste 'a' byteweise auf die Botschaft 100 kopiert und in Abständen von 10 ms in Blöcken von 8 Bytes auf den Bus gelegt. Durch Drücken der Taste 'r' wird die Datei an den Anfang zurückgesetzt und dann erneut über den Bus gesendet.

```
variables {
   message 100 msg = {DLC=8};
   char filename[50] = "test.dat";
   long handle;
   msTimer ReadNextBlock;
}

on start {
   // open data file on start of measurement
   handle = seqFileLoad(filename);
   if (handle <= 0) {
      write ("error opening file");
      write("error code = %d", handle); stop();
   }
   else {
      write ("file %s opened",filename);
      write ("Press key a to start reading the file");
   }
}

on stopMeasurement {
   // close data file on stop of measurement
   if (handle > 0) {
      if (seqFileClose(handle)==0) write("closed");
      else write ("error closing file");
   }
}

// start file transmission
on key 'a' { setTimer(ReadNextBlock,10); }

// reset file transmission
on key 'r' {
   seqFileRewind(handle);
   setTimer(ReadNextBlock,10);
}

// transmit data file in blocks of 8 bytes
on timer ReadNextBlock {
  int n;
  char buffer[8]= "       ";
  n = seqFileGetBlock(buffer,8,handle);
  msg.byte(0)= buffer[0]; msg.byte(1)= buffer[1];
  msg.byte(2)= buffer[2]; msg.byte(3)= buffer[3];
  msg.byte(4)= buffer[4]; msg.byte(5)= buffer[5];
  msg.byte(6)= buffer[6]; msg.byte(7)= buffer[7];
  output(msg);
  if (n >= 0) setTimer(ReadNextBlock,10);
  else write("No character read");
}
```

3.7.12 String-Funktionen

long atol(char s[])

Funktion:

Die Funktion wandelt den String *s* in eine LONG Zahl um. Die Zahlenbasis ist dezimal. Beginnt der String mit 0x, wird als Basis 16 verwendet. Führende Leerzeichen werden überlesen.

void ltoa(long val, char s[], long base)

Funktion:

Die Zahl *val* wird in einen String *s* umgewandelt. *base* gibt hierbei die Zahlenbasis zwischen 2 und 36 an. *s* muß groß genug sein, die umgewandelte Zahl aufzunehmen!

long snprintf(char dest[], long len, char format[], ...)

Funktion:

Die Funktion entspricht der C-Funktion `sprintf()`. Zusätzlich gibt der Parameter *len* die maximale Länge des Array *dest* an. Die Gesamtlänge des Ergebnisstrings darf 100 nicht überschreiten!

Funktionsergebnis ist die Anzahl der geschriebenen Zeichen. Der Format-String hat die gleiche Bedeutung wie bei der Funktion **`write()`** und ist dort beschrieben.

long strlen(char s[])

Funktion:

Das Funktionsergebnis ist die Länge des Strings *s*.

strncat(char dest[], char src[], long len)

Funktion:

Die Funktion hängt *src* an *dest* an. *len* gibt die maximale Länge von *src* und *dest* an. Die Funktion stellt eine abschließende '\0' sicher. Es werden also maximal *len*-1 Zeichen kopiert.

strncpy(char dest[], char src[], long len)

Funktion:

Die Funktion kopiert *src* nach *dest*. *len* gibt die maximale Länge von *src* und *dest* an. Die Funktion stellt eine abschließende '\0' sicher. Es werden also maximal *len*-1 Zeichen kopiert.

long strncmp(char s1[], char s2[], long len)

Funktion:

Die Funktion vergleicht *s1* mit *s2* auf maximal *len* Zeichen. Bei Gleichheit ist das Funktionsergebnis 0. Ist *s1* kleiner *s2* ist das Ergebnis -1, ansonsten +1.

3.7.13 Sprachunterstützung und Debugging

long elCount(...)

Funktion:

Ermittlung der Anzahl der Elemente eines Arrays.

Parameter:

Array beliebigen Typs

Rückgabe:

Anzahl der Elemente

Beispiel:

```
void bsp(int ar[])
{
   int i;
   for(i=0; i < elCount(ar); i ++)
      ...
}

void bsp2(byte ar[][])
{
   int i, j;
   for(j=0; j < elCount(ar); j ++ )
      for(i=0; i<= elCount(ar[j]); i ++ )
         ...
}
```

runError(long err, long)

Funktion:

Auslösung eines Laufzeitfehlers. Gibt im Write-Fenster eine Fehlermeldung unter Angabe der Fehlernummer und der übergebenen Zahl aus und bricht anschließend die Messung ab.

Parameter:

Zahlen, die im CANalyzer als Hinweis dargestellt werden. Die Werte unter 1000 sind für interne Zwecke reserviert. Der zweite Parameter ist für zukünftige Erweiterungen reserviert.

Beispiel:

```
if (Drehzahl < 0) runError(1001,1);
```

fileName()

Funktion:

Ausgabe des CAPL-Programmnamens im Write-Fenster. Hilfreich für Debug-Zwecke.

Beispiel:

```
fileName();
```

3.8 Beispiele

Um die folgenden Beispiel nachzuvollziehen, beginnen Sie am bestem immer mit einer neuen, leeren Konfiguration (Menüeintrag *Datei|Konfiguration neu*). Ein CAPL-Programm läuft immer in einem CAPL-Programmblock ab, den Sie an allen Hot-Spots im Meßaufbau bzw. in CANoe auch direkt am Bussymbol des Simulationsaufbaus einfügen können.

Hinweis: CAPL-Programme, die Botschaften senden sollen, *müssen* im CANalyzer immer im Sendezweig des Meßaufbaus bzw. in CANoe im Simulationsaufbaufenster einfügt werden. Das grafische Menü erlaubt es Ihnen zwar, CAPL-Programme auch im Meßaufbau direkt vor den Analyseblöcken einzufügen. Da der Datenfluß im Meßaufbau jedoch von links nach rechts gerichtet ist, gelangen Botschaften, die von dort aus gesendet werden, nur in die rechts liegenden Analyseblöcke und nicht auf den CAN-Bus. Sie sollten daher hier nur CAPL-Programme für die Analyse einfügen.

Ordnen Sie den CAPL-Programmblöcken, die Sie in den Meß- bzw. Simulationsaufbau eingefügt haben, dann im Kontextmenü einen Dateinamen zu, z.B. `EXAMPLE1.CAN`. Über den Menüpunkt *Bearbeiten* öffnen Sie den CAPL-Browser (vgl. Abschnitt 3.4), mit dem Sie die Beispielprogramme eingeben und compilieren können. Nach dem Compilieren können Sie die Programme sofort testen, wenn Sie die Messung starten.

Achten Sie darauf, daß Sie vor dem Messungsstart die CAN-Controller richtig parametrieren. In jedem Fall müssen Sie die Busparameter richtig einstellen. Um die Beispielprogramme nachzuvollziehen, sollten Sie die beiden CAN-Controller auf Ihrer Karten mit dem mitgelieferten Kabel verbinden („kurzschließen“). Sie haben so einen realen CAN-Bus mit zwei Stationen, die Sie beiden über den CANalyzer vollständig kontrollieren können. Denken Sie daran, in diesem Fall die Busparameter beider Controller genau gleich einzustellen.

3.8.1 Senden von Botschaften

Dieses Beispiel behandelt das Senden von Botschaften. Es wird zunächst gezeigt, wie periodisch gesendet wird. Dann wird das Beispiel durch Senden auf Tastendruck und durch Senden als Reaktion auf den Empfang einer Botschaft ausgebaut.

Aufgabenstellung:

Es sollen Botschaften mit dem Identifier 180 periodisch alle 100 msec gesendet werden. Die ersten beiden Datenbytes sollen als 16-bit Größe interpretiert werden und nach jedem Senden um 10 inkrementiert werden.

Lösung:

Es wird zunächst eine globale Variable mit dem Namen **sendTimer** vom Typ **msTimer** (Timer auf Millisekundenbasis) deklariert. Dies geschieht im Browser in der rechten oberen Pane:

```
variables {
   msTimer sendTimer; /* Timer für period. Senden */
}
```

Damit der so erzeugte Timer aktiv wird, muß dieser in der Start-Event-Prozedur aufgezogen werden. Zur Eingabe der Start-Event-Prozedur wählen Sie im Browser in der linken oberen Pane der Ereignistyp **System.** Im Kontextmenü der darunterliegenden Pane wählen Sie dann eine neue **Start**-Ereignisprozedur, die beim Messungsstart aufgerufen wird.

Als einzige Aktion beim Messungsstart wird **sendTimer** auf 100 msec aufgezogen.

```
on start {
   setTimer(sendTimer,100);  /* Timer einplanen */
}
```

Um festzulegen, was beim Ablauf des Timers geschehen soll, richten Sie jetzt eine Timer-Event-Prozedur ein. Wählen Sie dazu in der linken oberen Browser-Pane der Ereignistyp **Timer** aus und legen Sie in der Pane darunter eine Prozedur dieses Typs an. Dadurch erscheint in der rechten unteren Pane das Skelett einer Timer-Event-Prozedur, bei der Sie vor der öffnenden Prozedurklammer den Name des betreffenden Timers eintragen können, in diesem Fall also **sendTimer**.

Innerhalb der Prozedurklammer wird zuerst die lokale Variable **sendMsg** vom Typ **message** mit dem Identifier 180 deklariert. Die Variable **sendMsg** wird mit der Datenlänge 2 und mit dem Dateninhalt 0 initialisiert.

Der erste Befehl in der Event-Prozedur ist das erneute Aufziehen des Timers **sendTimer** auf 100 msec. Dadurch wird diese Event-Prozedur periodisch alle 100 msec durchlaufen. Dann wird die Botschaft **sendMsg** durch die Funktion **output()** an den Ausgang des Programmblocks ausgegeben. Da dort der Sendeblock anschließt, wird diese Botschaft gesendet. Nach dem Senden wird der Inhalt des ersten Datenwortes um 10 erhöht.

Nach dem Compilieren dieser Event-Prozedur erscheint im Prozedurfenster der Name **sendMsg**. Er kann für weitere Handhabung direkt selektiert werden.

```
on timer sendTimer {

   message 180 sendMsg = { DLC = 2 };

   // Timer period. einplanen
   setTimer(sendTimer,100);
   output(sendMsg); // senden
   // Datenwort um 10 erhöhen
   sendMsg.word(0) = sendMsg.word(0) + 10;
}
```

Über den Hauptmenüpunkt *Start|Start* wird das Programm gestartet. Die gesendeten Botschaften können im Trace- und im Statistikfenster beobachtet werden. Es wird beliebig lange gesendet, bis die Messung durch Drücken der *<ESC>*-Taste beendet wird.

Lösung der Aufgabe mit symbolischer Datenbasis

Die verwendete **on timer**-Prozedur löst die gestellte Aufgabe, indem sie direkt auf Botschaftsidentifier und Datenbytes zurückgreift. Dies ist zulässig, und wird von der Aufgabenstellung auch nahegelegt. Denoch wird das CAPL-Programm durch die Vermischung von Applikation (Ausgabe eines Zählwertes) und busnahen Informationen (Botschaft, Datenbytes) unflexibel und schlecht wartbar: Wenn in einer Aufgabenerweiterung beispielsweise der Zähler nun auch Werte bis 100000 erfassen soll, reichen die ersten beiden Datenbytes der Botschaft nicht mehr aus. Das CAPL-Programm muß also modifiziert werden.

Um Applikationsaufgabe und Datenübertragung besser auseinanderzuhalten, bieten Ihnen der CANalyzer und CANoe die Möglichkeit, die CAN-Daten symbolisch zu beschreiben. Statt mit Identifiern und Datenbytes arbeiten Sie dann in CAPL mit symbolischen Namen. Aufgabenerweiterungen, die eine Änderung der Datenübertragungsschicht erforderlich machen, lassen sich so weitgehend durch eine Modifikation dieser symbolischen Beschreibung handhaben. Eine Änderung der Applikation, d.h. in diesem Fall des CAPL-Programms, ist dann nicht mehr notwendig.

Eine Datenbasis für obige Aufgabenstellung sollte eine symbolische Beschreibung der Botschaft mit dem Identifier 180 und ein Signal *Zaehler* enthalten, das die ersten beiden Bytes dieser Botschaft umfaßt:

Botschaftsname	StateInfo
Botschafts-Id:	180
Anzahl Bytes	2

Signalname	Zaehler
Signaltyp	Standardsignal
Startbit	0
Anzahl Bits	16

Damit läßt sich die **on timer**-Prozedur zur Lösung der Aufgabe folgendermaßen schreiben:

```
on timer sendTimer {

   message StateInfo sendMsg; // DLC aus Datenbasis

   setTimer(sendTimer,100);
   output(sendMsg);
   // Zaehler um 10 erhöhen
   sendMsg.Zaehler = sendMsg.Zaehler + 10;
}
```

Beachten Sie, daß zur Typdefinition der zu sendenden Botschaft sowohl das Schlüsselwort **message** als auch der Botschaftsname **StateInfo** gehören. Erst danach folgt der Variablenname **sendMsg**. Um in Ereignisprozeduren auf Botschaften zu reagieren, verwenden Sie den symbolischen Botschaftsnamen aus der Datenbasis und keinen Variablennamen. Schreiben Sie also beispielsweise **on message StateInfo {}**, um auf das Auftreten der Botschaft mit dem Identifier 180 zu reagieren.

Erste Aufgabenerweiterung:

Zusätzlich soll jetzt bei jedem Tippen der Leertaste die Botschaft mit dem Identifier 210 gesendet werden. Als Dateninhalt im Byte 3 soll angegeben werden, wie oft bereits gesendet wurde.

Lösung:

Um festzulegen, was beim Drücken der Leertaste geschehen soll, richten Sie die entsprechende Key-Event-Prozedur ein. Der Name von Key-Event-Prozeduren ist entweder das entsprechende ASCII-Zeichen in Hochkommata oder ein symbolischer Name (vgl. Abschnitt 3.5.4). Wählen Sie in der linken oberen Browser-Pane den Ereignistyp **on key** aus. Im Skelett der neuen Key-Event-Prozedur ersetzen Sie den Text **<newKey>** durch das Leertastenzeichen **' '**.

Innerhalb der Prozedurklammer wird zuerst die lokale Variable **spaceMsg** deklariert. Beim Durchlaufen der Prozedur wird diese Botschaft gesendet. Als Zähler für die Sendehäufigkeit wird direkt das 3. Datenbyte verwendet.

```
on key ' ' {
  message 210 spaceMsg = {DLC=4, DIR=TX, byte(2)=0};

  output(spaceMsg);           /* senden */
  spaceMsg.byte(2) = spaceMsg.byte(2)++; /* mitzählen */

}
```

Zweite Aufgabenerweiterung:

Zusätzlich soll jetzt auf den Empfang der Botschaft mit dem Identifier 120 durch das Aussenden der Botschaft 121 reagiert werden. Für Testzwecke soll außerdem eine Meldung am Bildschirm ausgegeben werden.

Lösung:

Um festzulegen, was beim Empfang einer bestimmten Botschaft geschehen soll, richten Sie die entsprechende Message-Event-Prozedur ein. Legen Sie dazu eine Event-Prozedur vom Typ **on message** an. Im Skelett der Prozedur ersetzten Sie den Text <newMessage> durch den Identifier, auf den reagiert werden soll: **120**.

Innerhalb der Prozedurklammer wird zuerst die lokale Variable **responseMsg** deklariert. Beim Durchlaufen der Prozedur wird der Empfang der Botschaft 120 am Bildschirm über die **write()**-Funktion gemeldet. Anschließend wird die Antwortbotschaft gesendet.

```
on message 120 {

  message 121 responseMsg = { DLC = 0, DIR = TX };

  write("Msg 120 received");
  output(responseMsg); /* Antwort senden */
}
```

In der Event-Prozedur könnte mit folgender Bedingung noch untersucht werden, ob die Botschaft tatsächlich empfangen oder vielleicht gesendet wurde:

```
if (this.dir == RX) { ... }
```

Da auf diese Abfrage verzichtet wird, reagiert die Event-Prozedur auf empfangene und gesendete Botschaften gleich.

Der Test dieser Event-Prozedur ist nur möglich, wenn tatsächlich eine Botschaft mit dem Identifier 120 empfangen (oder gesendet) wird. Diese kann durch eine fremde Busstation oder von CANalyzer bzw. von CANoe selbst generiert werden. Ein schneller Test ist z.B. dadurch möglich, daß anstelle des Identifiers 120 der Identifier 210 gewählt wird, der, wie oben programmiert, beim Drücken der Leertaste gesendet wird.

3.8.2 Gateway-Anwendung

Im nachfolgenden Beispiel soll eine Gateway-Anwendung programmiert werden, bei der zwei Busse über den CANalyzer bzw. über CANoe verbunden werden. Dazu muß selbstverständlich eine PC-Karte mit zwei CAN-Controllern eingesetzt werden, die über getrennte Busanschlüsse herausgeführt werden. Im Gegensatz zum vorhergehenden Beispiel, muß hier bei der CAPL-Programmierung genau spezifiziert werden, über welchen Bus Botschaften gesendet bzw. empfangen werden sollen.

Aufgabenstellung:

Es soll ein Gateway zwischen zwei Bussen programmiert werden, das prinzipiell alle Botschaften in beiden Richtungen ungehindert passieren läßt. Wenn allerdings die Botschaft mit dem Identifier 34 von Bus 1 nach Bus 2 übertragen wird, soll sie „verfälscht“ werden, indem Byte 4 auf 0 gesetzt wird.

Lösung:

Es wird ein Netzknoten im Sendezweig (CANalyzer) bzw. im Simulationsaufbau (CANoe) eingefügt, dem das nachfolgend beschriebene CAPL-Programm zugeordnet wird.

Das Verfälschen der Botschaft 34 beim Übergang von Bus 1 nach Bus 2 wird wie folgt ausgeführt:

```
on message CAN1.34 {   /* beim Empfang der Botschaft 34
                          über Bus 1 */

  message CAN2.34 sendMsg;
  if (this.dir!=RX) return; // reagiere nur auf Empfang
  sendMsg=this; // kopiere Empfangs in Sendebotschaft
  sendMsg.byte(4) = 0; // verfälschen
  output(sendMsg); // senden
}
```

Für die restlichen Botschaften wird eine Bridge-Funktion realisiert: Alle Botschaften von Bus 1 werden zu Bus 2 übertragen und umgekehrt.

```
on message CAN1.* {
  // durch die Angabe '*' werden alle Botschaften
  // bezeichnet, die in keiner anderen Prozedur
  // verarbeitet werden.
   message CAN2.* sendMsg;
   if (this.dir!=RX) return;
   sendMsg = this;
   output(sendMsg);
}

on message CAN2.* {
  message CAN1.* sendMsg;
  if(this.dir!=RX) return;
  sendMsg = this;
  output (sendMsg);
}
```

3.8.3 CAPL-gesteuerte Triggerung

In diesem Abschnitt wird in einem sehr einfachen Beispiel gezeigt, wie die Triggerung der Datenaufzeichnung auf Datei (Logging) über ein CAPL-Programm ausgelöst werden kann.

Aufgabenstellung:

Der Anwender beobachtet das System und möchte die Triggerung über einen Tastendruck (Taste 't') auslösen.

Lösung:

Bei der Parametrierung des Logging-Blocks wird als *Triggertyp* CAPL gewählt. Damit wird die Dateiaufzeichnung durch die CAPL-Funktion **`trigger()`** ausgelöst werden. Beachten Sie, daß die Funktion das Logging triggert, wenn sie sich in einem CAPL-Programm vor dem Logging-Zweig oder aber im Sendezweig des CANalyzers bzw. im Simulationsaufbau von CANoe befindet.

Das CAPL-Programm besteht aus einer einzigen Event-Prozedur für die Taste 't' in der die Triggerung ausgelöst wird.

```
on key 't' { // beim Tastendruck der Taste 't'
             // (ohne Shift)

  trigger(); // Triggerung auslösen
}
```

Natürlich kann ein CAPL-Programm zur Triggerung auch wesentlich komplizierter aussehen. So kann z.B. durch das Hinzufügen einer Message-Event-Prozedur die Triggerung zusätzlich ausgelöst werden, wenn eine bestimmte Botschaft mit bestimmtem Dateninhalt empfangen wurde.

Da globale Variablen (Flags) eingeführt werden können, können Sie auch Triggerbedingungen formulieren, die von der Kombination verschiedener Ereignisse abhängig sind.

3.8.4 Verwendung der Symboldatenbasis

In diesem Beispiel wird gezeigt, wie auf symbolische Signale zugegriffen wird. Hierzu muß eine Projektdatenbasis in den CANalyzer bzw. in CANoe[6] geladen werden (siehe CANdb).

Die Datenbasis muß dabei eine Botschaft mit dem symbolischen Namen `ioData` enthalten, auf der die Drehzahlinformation im Signal `N` kodiert ist. Erforderlich ist außerdem eine weitere Botschaft mit dem Namen `Motordaten` und dem Signal für das Nullmoment `NM`.

Aufgabenstellung:

Auf CAN2 liegt über ein Analog I/O-Modul eine skalierte Drehzahlinformation **`N`** an. Aus dieser soll ein Nullmoment-Signal **`NM`** berechnet werden und periodisch auf CAN1 ausgegeben werden.

Lösung:

Der Wert des Signals **`N`** wird ermittelt, in **`NM`** umgerechnet und in einer Variablen **`nullMoment`** zwischengespeichert. In der Timer-Funktion wird auf deren Wert zugegriffen und die periodische Ausgabe erzeugt:

[6] Beim Arbeiten mit CANoe ist die Verwendung einer Datenbasis sogar obligatorisch.

```
variables {
   int nullMoment = 0;     /* berechneter Wert */
   int c1 = 10, c2 = 20;   /* Umrechnungskonstanten */
   msTimer t1;             /* Timer */
}
```

```
on start {
   setTimer(t1,100);   /* Aufziehen des Timers */
}
```

```
on message CAN2.ioData {
   nullMoment=this.N*c1+c2; /* lineare Transformation */
}
```

```
on timer t1 {
   message CAN1.Motordaten m; /* Datenträger */
   setTimer(t1,100);          /* Aufziehen des Timers */
   m.NM = nullMoment;         /* Eintragen des aktuellen
                                 Wertes */
   output(m);                 /* Aussenden auf CAN1 */
}
```

Aufgabenerweiterung:

Die Berechnung des Nullmoments soll tabellengesteuert erfolgen.

Lösung:

Es wird eine Tabelle mit den Ergebniswerten angelegt. Der Zugriff erfolgt indiziert, wobei der Wertebereich überprüft wird.

```
variables {
   int nullMoment = 0;       /* berechneter Wert */
   int values[20] = {        /* Tabelle */
        0,0,3,8,15,24,35,48,63,80,
        99,118,140,164,190,218,220,
        218,210,195
       }
   msTimer t1;               /* timer */
}
```

```
on message CAN2.ioData {
   int index;                  /* Hilfvariable */
   index = this.N;             /* Wert als Index */
   if (index < 0) index = 0;   /* Bereichsprüfung */
   else if (index>=20) index = 19;
   nullMoment = values[index]; /* Umrechnung
                                  mit Tabelle */
}
```

Eine typische Aufgabenerweiterung wäre es, die Berechnung als Interpolation über ein Datenfeld auszuführen.

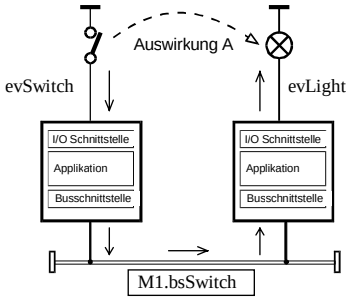
Hinweis: Beachten Sie, daß bei dieser Lösung Informationen über die CAN-Protokollschicht in das CAPL-Programm einfließen: Die CAN-Controller-Nummern sind in diesem Lösungsvorschlag Bestandteil des CAPL-Programms. Wollte man die CAN-Controller der Karte anders anschließen, müßte man also den Programmcode modifizieren. Wie Sie diese unerwünschten Abhängigkeiten der CAPL-Applikation von der konkreten Konfiguration vermeiden können, wird in Abschnitt 4.3.1 beschrieben.

3.8.5 Erstellen eines Netzknotenmodells mit CAPL in CANoe

Um das funktionale Busverhalten von Netzknoten zu modellieren, verfügt CANoe über „Umgebungsvariablen“, mit denen Ereignisse und Zustände der Systemumgebung (äußerer Druck, Temperatur, Schalterstellungen, ...) beschrieben werden. Sie können diese Zustände, d.h. die Werte der Umgebungsvariablen auf frei gestaltbaren Bedienpanels sowohl beobachten als auch gezielt verändern.

Zum Arbeiten mit Umgebungsvariablen steht Ihnen in CAPL der Ereignisprozedurtyp **`on envVar`** zur Verfügung. Zum Lesen und Schreiben der Umgebungsvariablenwerte verwenden Sie die CAPL-Funktionen **`getValue()`** und **`putValue()`**.

Diese Sprachmittel und der symbolische Zugriff auf die verschiedenen in der Datenbasis definierten Variablen ermöglichen die einfache prototypische Erstellung von Modellen der Knoten. Das folgende Beispiel zeigt jeweils eine Ereignisprozedur für die Änderung einer Umgebungsvariablen, den Empfang einer Busbotschaft und für den Ablauf eines Timers. Die Datenbasis, die dem Beispiel zugrunde liegt, enthält dabei die Umgebungsvariablen `evSwitch` und `evLight`, die die Position eines Lichtschalters bzw. den Zustand eines Lämpchens beschreiben, die Botschaft `M1` mit dem Signal `bsSwitch`, die die Schalterposition kodiert und anderen Netzknoten zur Verfügung stellt.



Das erste CAPL-Programm gehört zu einem Netzknoten, an dessen Peripherie sich ein Schalter befindet. Das Programm erfaßt bei Änderung der Schalterposition den neuen Schaltwert und gibt diesen unmittelbar auf den Bus aus:

```
// Reaktion auf die Änderung der Umgebungsvar. evSwitch
on envVar evSwitch {

   // Deklaration einer zu versendenden CAN-Botschaft
   message M1 msg;

   // Auslesen des Wertes des Lichtschalters,
   // Zuweisung an das Bussignal bsSwitch
   msg.bsSwitch = getValue(this);

   // Ausgabe d. Botschaft auf Bus (spontanes Senden)
   output(msg);
}
```

Ein zweiter Netzknoten reagiert auf diese Botschaft. Das CAPL-Programm liest den Wert des Bussignals für die Schalterposition und schaltet daraufhin das Lämpchen an seiner Peripherie an oder aus. Beachten Sie, daß der Wert des Schalters allein über den Signalwert am Bus erfaßt wird. Der Wert der Umgebungsvariablen evSwitch ist diesem CAPL-Programm nicht bekannt. Die Kommunikation zwischen beiden Knoten erfolgt also ausschließlich über den CAN-Bus:

```
// Reaktion auf den Empfang der CAN-Botschaft M1
on message M1 {
   // Auslesen eines Bussignals und
   // Setzen der Umgebungsvariablen
   putValue(evLight, this.bsSwitch);
}
```

Beachten Sie, wie einfach und direkt sich dieses Modell eines einfachen verteilten Systems in CAPL modellieren läßt, wobei der Datenbasis hier eine zentrale Bedeutung zukommt.

Die Kommunikation zweier Netzknoten über den CAN-Bus mittels spontaner Emission, d.h. direktem Aussenden einer CAN-Nachricht bei Zustandsänderungen ist jedoch nicht die einigen Möglichkeit, um in CAPL das Busverhalten eines Netzknotens zu modellieren. Mit CAPL-Sprachmitteln wie z.B. den Timern lassen sich ebenso leicht auch zyklische Sendeprotokolle realisieren.

Hinweis: Beachten Sie, daß die Ereignisprozedur **`on envvar ... {}`** nicht synchron zur entsprechenden Funktion **`putValue()`** aufgerufen wird, sondern zeitlich entkoppelt ist. Zwischen beiden Funktionen können also durchaus andere Ereignisprozeduren aufgerufen werden.

4 Spezielle Themen

4.1 Systembeschreibung

4.1.1 Übersicht über die Programme

Zum CANalyzer gehören drei ausführbare Programme:

1. Mit dem Datenbasis-Editor erstellen oder modifizieren Sie die Datenbasen (`*.DBC`), welche die symbolische Information für den CANalyzer enthalten. Dazu gehören symbolische Namen für Botschaften und Signale sowie die Umgebungsvariablen.
2. Im CAPL-Browser erstellen Sie die CAPL-Programme für den Sende- und Analysezweig des Messaufbaus. Statt mit Botschafts-IDs und Datenbytes zu arbeiten, können Sie mit Hilfe der Datenbasis auch mit Botschafts- und Signalnamen arbeiten.
3. Das CANalyzer-Hauptprogramm zum Messen und Stimulieren von CAN-Systemen. Ordnen Sie jeder CANalyzer-Konfiguration mit *Datei|Datenbasis* eine Datenbasis zu.

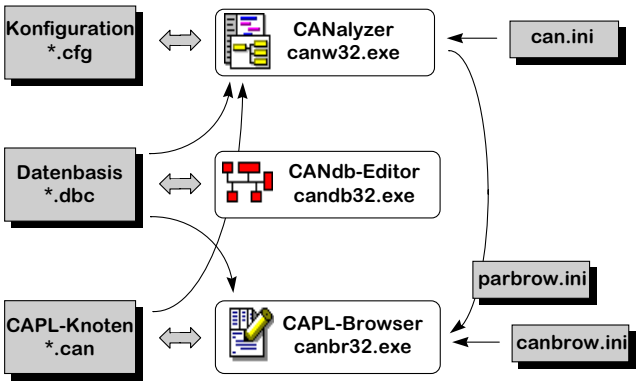


Abbildung 64: CANalyzer Systemübersicht

Startoptionen für den CANalyzer und den Browser werden in den entsprechenden INI-Dateien bereitgestellt. Falls Sie den Browser aus dem Messaufbau des CANalyzers starten, wird eine temporären Datei `PARBROW.INI` mit den richtigen Startoptionen automatisch generiert und dem Browser übergeben.

4.1.2 CANalyzer Architektur

Im Verlauf einer Messung registriert die PC-Einsteckkarte CAN-Botschaften auf dem Bus und reicht diese auf den im Datenflußdiagramm angegebenen Pfaden an die ganz rechts stehenden Auswerte- und Analyseblöcke weiter. Zu diesem Zweck arbeiten während einer Messung zwei Programmodule des CANalyzers eng

zusammen. Zunächst holt die CANalyzer-Echtzeitbibliothek (`CANRT.DLL`) die an der Karte anfallenden Informationen ab und legt diese in einem Ringpuffer ab.

Diese Daten werden in einem zweiten Schritt vom eigentlichen Hauptprogramm (`CANW.EXE` bzw. `CANW32.EXE`) ausgelesen und in den Funktionsblöcken auf der rechten Seite des Datenflußplans ausgewertet.

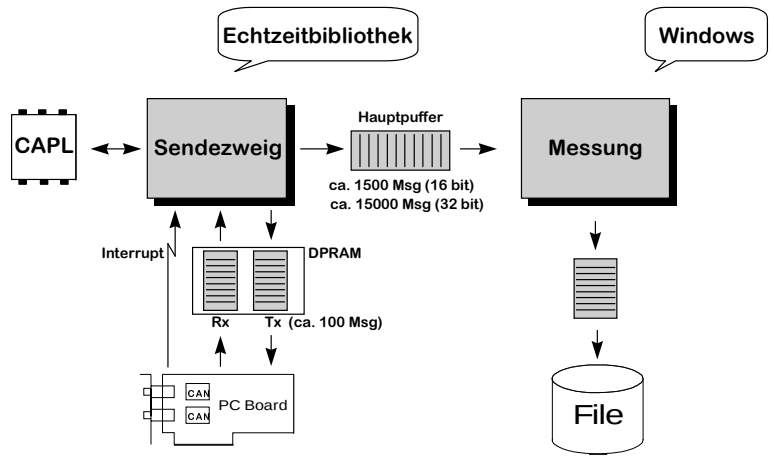


Abbildung 65: Interner Aufbau des CANalyzers

Sie können den Datenfluß in beiden Programmodulen beeinflussen, indem Sie Funktionsblöcke in den Meßaufbau einfügen. Das Echtzeitmodul umfaßt dabei den PC-Kartenblock, alle Funktionsblöcke zwischen Kartenblock und Sendezweig sowie alle Blöcke im Sendezweig selber. Durch alle weiteren Funktionsblöcke konfigurieren Sie den Datenfluß im Auswertezweig.

Wenn Sie Blöcke in den Echtzeitzweig des CANalyzers einfügen, sollten Sie darauf achten, daß diese nicht zuviel Rechenzeit verbrauchen, damit die Reaktionszeiten des System nicht verlängert werden. Außerdem dürfen Sie in CAPL-Programmen von hier nur mit speziellen Vorsichtsmaßnahmen auf Dateien zugreifen (vgl. Abschnitt 3.7.10)

4.2 Der CANalyzer im Last- und Überlastbetrieb

4.2.1 Verhalten in Lastsituationen

Bei hoher Busbelastung kann die Rechenleistung ihres PCs unter Umständen nicht ausreichen, um die umfangreicher Auswerte- und Anzeigefunktionen (Statistikfenster mit Statistikprotokoll, Daten- und Grafikfenster mit vielen Signalen, Trace-Fenster im Ausgabemodus Zeitlich folgend, ...) gleichzeitig auszuführen. Um einen drohenden Datenverlust zu verhindern, verfügt das Programm daher über Mechanismen um Lastsituationen zu erkennen und zu bewältigen.

Falls die Rate der von der Karte registrierten Botschaften so hoch ist, daß der CANalyzer mit der Verarbeitung nicht mehr folgen kann, läuft der Ringpuffer zwischen der Echtzeitbibliothek und der CANalyzer Applikation voll. Der CANalyzer erkennt dies am Überschreiten einer „High-Water-Mark“-Grenze (vgl. Abbildung 66) und schaltet automatisch in den Lastbetrieb, bei dem die Anzeigefunktionen reduziert werden, um mehr Zeit für die interne Datenverarbeitung bereit zu stellen.

So unterbricht das Trace-Fenster, die Botschaftsdarstellung im Lastbetrieb kurzzeitig, um anderen Analyseblöcken mehr Rechenleistung zur Verfügung zu stellen. Damit werden aber während der Messung u.U. nicht mehr alle Botschaften in Fenster dargestellt. Sie erkennen diese Lastsituation während der Messung durch ein Ausrufezeichen (!) in der ersten Spalte des Fensters. Obwohl nicht mehr alle Botschaften dargestellt werden, gehen jedoch keine Daten verloren. Nachdem Sie die Messung beendet haben, stehen Ihnen die vollständigen Informationen im Trace-Fenster, im Grafikfenster wie auch im Logging zur Verfügung.

4.2.2 Verhalten bei Datenverlust

Falls der Ringpuffer dennoch überlaufen sollte, werden Sie als Anwender von diesem Datenverlust umgehend informiert:

Im Trace-Fenster erscheint dazu während des Überlaufs zunächst das Überlastzeichen ‘@’ vor der betreffenden Botschaft. Sobald der Ringpuffer die Daten wieder fassen und korrekt an das Windows-Hauptprogramm weiterleiten kann, wird im Trace-Fenster anstelle des ‘@’-Zeichens ein ‘*’-Zeichen angezeigt.

Dies weist Sie darauf hin, daß der Ringpuffer zwar momentan nicht mehr überläuft, die Daten unmittelbar nach der betreffenden Zeile aber verloren sind. Sie beobachten dieses Verhalten am besten im Ausgabemodus *Zeitlich folgend*, da im feststehenden Modus das ‘*’-Zeichen am Zeilenanfang überschrieben wird, sobald eine neue Botschaft nach Beendigung der Überlast korrekt in die betreffende Zeile geschrieben wird.

Analog zum Trace-Fenster wird auch im ASCII-Logging ein aufgetretener Datenverlust registriert. In der Zeile, nach welcher der Datenverlust eingetreten ist, erscheint ebenfalls das Zeichen ‘*’. Im Konfigurationsdialog der Logging-Datei haben Sie mit der Option *Lost Data Messagebox* zusätzlich die Möglichkeit, sich bei Messungsende einen Datenverlust in einem eigenen Meldungsfenster anzeigen zu lassen.

Auch das Busstatistikfenster zeigt Ihnen während der Überlastsituation mit dem ‘@’-Zeichen einen Datenverlust an. Beachten Sie jedoch, daß die Anzeige der Buslast und der empfangenen Botschaften trotzdem weiter korrekt funktionieren, da diese Informationen bereits von der Schnittstellenkarte zur Verfügung gestellt werden und daher nicht erst im Hauptprogramm berechnet werden müssen. Die Busstatistikanzeige ermöglicht es Ihnen daher, den Umfang des Datenverlustes abzuschätzen.

4.2.3 Füllstandsanzeige

Um den Ringpuffer zwischen der Echtzeitbibliothek und dem Windows-Hauptprogramm genauer beobachten zu können, besitzt dieser eine Füllstandsanzeige. Sie können sich die Zustände dieser Anzeige im Write-Fenster

anzeigen lassen. Setzen Sie dazu in der Datei CAN.INI im Abschnitt [Environment] mindestens den Wert WriteLevel = 3. Beim Überlaufen des Ringpuffers erscheint dann im Write-Fenster die Meldung „Lastübergang: NORMAL->DATA LOST“ und setzt Sie vom Datenverlust in Kenntnis. Nachdem die Überlastsituation beendet ist (etwa, nach einem kurzfristigen Burst auf dem Bus) werden Sie ebenfalls informiert, sobald die Normalsituation wiederhergestellt ist. Sie erkennen dies im Write-Fenster an der Meldung „Lastübergang: QUEUE HIGH -> NORMAL“.

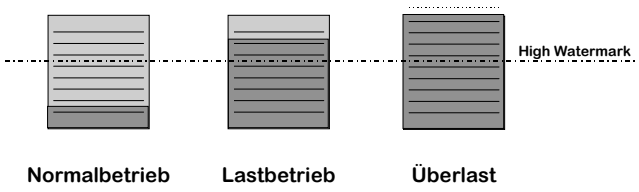


Abbildung 66: Füllstand des Ringpuffers

Sie können zu Testzwecken selber Überlastsituationen provozieren, indem Sie mit der Maus die Titelzeile des Hauptfensters festhalten, bzw. das Fenster auf dem Bildschirm verschieben. Die Datenanzeige des Hauptprogramms bleibt dabei solange unterbrochen, bis Sie die Titelzeile wieder loslassen. Die Arbeit der Echtzeitbibliothek bleibt dagegen von diesen Aktionen unbeeinflußt. Sofern Botschaften auf dem Bus registriert werden, wird der Ringpuffer gefüllt, ohne daß die Daten vom Hauptprogramm bearbeitet werden könnten, und läuft schließlich über. Sobald Sie die Titelzeile loslassen, können Sie dann die Auswirkungen dieses Überlaufs in den entsprechenden Fenstern erkennen.

4.2.4 Konfigurationsmöglichkeiten bei hoher Buslast

Neben den automatischen Abschaltefunktion, die der CANalyzer in Lastsituationen aktiviert, können Sie folgende Analyseblöcke manuell in weniger rechenintensive Modi umschalten:

Trace-Fenster

Wählen Sie im Trace-Fenster den Ausgabemodus *Feste Position bei zyklischem Update*. Damit wird der Fensterinhalt nicht mehr bei jeder neu ankommenden Botschaft sondern nur noch zyklisch aktualisiert. Die Zykluszeit in Millisekunden stellen Sie in der Datei CAN.INI im Abschnitt [SYSTEM] mit CyclicUpdateTime = *nnnn* ein. Der Default beträgt 100 ms und ist für die meisten Rechner optimal.

Datenfenster

Falls Sie viele Signale im Datenfenster konfiguriert haben, wählen Sie hier den zyklischen Zeichenmodus (Eintrag *Timer konfigurieren* im Kontextmenü des Datenfensters) und geben Sie eine Zykluszeit von maximal 500 Millisekunden ein. Das Fenster wird dann ebenfalls nur noch zyklisch aktualisiert und kann damit Rechenleistung einsparen.

Grafikfenster

Falls Sie im Grafikfenster viele Signale konfiguriert haben, wählen Sie im Meßkonfigurationsdialog (Eintrag *Einstellungen* im Kontextmenü) einen relativ großen benutzerdefinierten Refresh (200 ms bis 2 s). Damit legen Sie fest, wie oft die Anzeige des Grafikfensters aktualisiert werden soll. Kleine Werte haben eine flüssige Darstellung des Signalverlaufs zur Folge, erfordern andererseits aber eine hohe Rechnerleistung und führen bei langsameren Rechnern eventuell zu Performanceproblemen, während hohe Werte den Rechenaufwand senken, aber zu einer unruhigeren und verspäteten Anzeige des Signalverlaufs führen.

Statistikfenster

Deaktivieren Sie das Statistikprotokoll im Kontextmenü des Statistikfensters, um Rechenleistung zu sparen. Im Konfigurationsdialog des Statistikblocks läßt sich ferner die Mittelungszeit wählen. Kurze Zeitintervalle erfordern einen hohen Rechenaufwand und haben u.U. stark schwankende Linien im Fenster zur Folge. Sehr lange Mittelungszeiten machen die Anzeige entsprechend träge aber auch weniger rechenintensiv.

Wenn Sie Blöcke in den Echtzeitzweig (vgl. Abschnitt 4.1.2) des CANalyzers einfügen, sollten Sie darauf achten, daß diese nicht zuviel Rechenzeit verbrauchen, damit die Reaktionszeiten des System nicht verlängert werden. Außerdem dürfen Sie in CAPL-Programmen von hier nur mit speziellen Vorsichtsmaßnahmen auf Dateien zugreifen (vgl. Abschnitt 3.7.10)

Andererseits kann es sinnvoll sein, bei hoher Busbelastung bereits im Echtzeitzweig eine Datenreduktion durchzuführen (etwa durch Einfügen eines Filterblocks), um den Analysezweig des CANalyzers zu entlasten.

Eine optimale Konfiguration des Meßaufbaus läßt sich nicht für alle Situationen im voraus bestimmen. Zyklisches Aktualisieren spart zwar Rechenzeit, führt aber auch zu einer schlechteren Darstellung der Informationen. Unter Umständen kann es sinnvoll sein, nicht benötigte Analysezweige im Meßaufbau ganz zu unterbrechen (Eintrag *Füge Unterbrechung* ein im Kontextmenü des Hot-Spots) bzw. die Datenmenge am Eingang des Meßaufbaus mit Filterfunktionen zu reduzieren. Sie können darüber hinaus versuchen, die Blöcke einzeln zwischen Echtzeitbibliothek und Analysezweig zu verschieben und das Verhalten des CANalyzers während eines erneuten Meßlaufs zu beobachten.

Um Botschaften gezielt aus dem Meßaufbau auszufiltern, stehen Ihnen Durchlaß- und Sperrfilter als einfügbare Funktionsblöcke zur Verfügung. Daneben bieten Ihnen die unterstützten PC-Einsteckkarten mit der Akzeptanzfilterung (Eintrag *Botschaften* im Kontextmenü des Kartensymbols im Meßaufbau) auch die Möglichkeit, bestimmte Botschaften bereits auf der Hardware auszufiltern und so zu verhindern, daß sowohl die Echtzeitbibliothek als auch das Windows-Hauptprogramm mit der Auswertung nicht benötigter Informationen belastet wird.

4.3 Arbeiten mit Datenbasen

Im symbolischen Modus des CANalyzers können Sie die CAN-Botschaften und Dateninhalte über symbolische Namen aus den zugeordneten Datenbasen

ansprechen. Die folgenden Abschnitte geben Ihnen Hinweise, worauf Sie achten müssen, wenn Sie beim Arbeiten mit dem CANalyzer Datenbasen verwenden.

4.3.1 Zuordnen von Datenbasen

Im Datenbasisauswahldialog, den Sie über den Menüeintrag *Datei|Datenbasis zuordnen* öffnen, bestimmen Sie, mit welchen Datenbasen Sie arbeiten wollen.

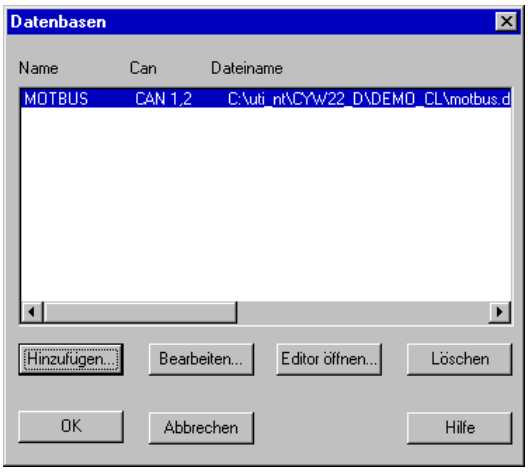


Abbildung 67: Dialog Datenbasis zuordnen

Unter *Hinzufügen...* können Sie über einen Dateiauswahldialog neue Datenbasen zuordnen. Ist in der Liste der Datenbasen keine Zeile markiert, so wird die neue Datenbasis am Ende hinzugefügt. Ansonsten wird über der markierten Zeile eingefügt. Um am Listenende einfügen zu können, läßt sich auch die leere Endzeile markieren.

In der Liste der Datenbasen werden in der ersten Spalte die Datenbasisnamen angezeigt, die Sie z.B. bei der Auflösung von Mehrdeutigkeiten als Namensqualifizierer verwenden. In der zweiten Spalte stehen die den Datenbasis zugeordneten CAN-Chip. In der letzten Spalte finden Sie den vollständigen Dateinamen der Datenbasis. Die Reihenfolge der Datenbasen in der Liste wird zur der Auflösung von Mehrdeutigkeiten bei symbolischen Namen und bei Botschafts-Identifiern herangezogen.

Durch Betätigen von der Schaltfläche *Bearbeiten...* kommen Sie in den Dialog *Datenbasis bearbeiten*. Hier können Sie den Namen der Datenbasis sowie die Zuordnung der Datenbasis zu einem CAN-Chip festlegen.

Der Name der Datenbasis wird folgendermaßen bestimmt: Enthält die Datenbasis das Attribut *DBName*, so wird als Name dessen Wert verwendet. Falls das Attribut nicht existiert, wird der Name aus dem Dateinamen abgeleitet. Sie können den Namen überschreiben. Datenbasisnamen müssen mit einem Buchstaben beginnen und dürfen weder Leer- noch Sonderzeichen enthalten.

Der Datenbasisname muß eindeutig sein. Ist ein automatisch bestimmter Name bereits vorhanden, so werden der Reihe nach die Namen CANdb1, CANdb2, etc. vergeben.

Sie können beide CAN-Controller (CAN1 bzw. CAN2) genau einer Datenbasis zuordnen. Alle weiteren Datenbasen sind beiden Controllern gemeinsam zugeordnet. Diese Zuordnung bestimmt den Defaultselektor für Botschaftsdefinitionen in CAPL.

4.3.2 Verwendung mehrerer Datenbasen

Bei großen Systemen kann es sinnvoll sein, die Beschreibung der Botschaften und Signale sowie die Umgebungsvariablen auf mehrere Teildatenbasen aufzuteilen. Auch beim Betrieb des CANalyzers an zwei Bussen liegt es nahe, jedes System durch eine eigene Datenbasis zu beschreiben.

Der CANalyzer unterstützt die gleichzeitige Verwendung mehrerer Datenbasen. Sie konfigurieren die Datenbasen, die Sie dem CANalyzer zuordnen möchten, mit dem Menüpunkt *Datei|Datenbasis zuordnen*. Sie können danach in allen Funktionsblöcken und in CAPL die symbolischen Bezeichner für Botschaften, Signale und Umgebungsvariablen aus allen Datenbasen verwenden. Tragen Sie dazu den symbolischen Namen in das entsprechende Eingabefeld ein. Eine Liste aller symbolischen Namen finden Sie in den Signalauswahldialogen, die Sie durch Betätigen der kleinen Schaltflächen neben den entsprechenden Eingabefeldern öffnen. Wählen Sie hier die gewünschten symbolischen Namen aus.

Wenn Sie mehr als eine Datenbasis verwenden, werden die Botschaften in den auf die erste folgenden Datenbasen mit dem Datenbasisnamen *qualifiziert*, d.h. den Botschaftsnamen wird der Datenbasisname gefolgt von zwei Doppelpunkten vorangestellt. Sie benötigen diese qualifizierten Namen jedoch nur um Mehrdeutigkeiten aufzulösen. Solange die symbolischen Namen in allen Datenbasen eindeutig sind, können Sie sowohl in allen Funktionsblöcken als auch beim Editieren von CAPL-Programmen auf die Qualifizierung der symbolischen Namen verzichten.

4.3.3 Auflösung von Mehrdeutigkeiten

Bei der Verwendung von mehreren Datenbasen sind bei der Verwendung symbolischer Namen prinzipiell Mehrdeutigkeiten möglich, die vom Programm aufgelöst werden müssen. Einerseits können Botschaften, die vom Bus kommen und über einen der beiden CAN-Controller vom Programm registriert werden, in zwei Datenbasen unterschiedliche symbolische Namen haben. Andererseits will der Anwender u.U. Funktionsblöcke oder Meßfenster mit unterschiedlichen Botschaften konfigurieren, die in verschiedenen Datenbasen den gleichen Namen tragen.

Mehrdeutigkeiten der ersten Art werden aufgelöst durch die Reihenfolge, in der Sie die Datenbasen in die Liste des Datenbasisauswahldialogs angegeben haben. Zusätzlich haben Sie die Möglichkeit, jedem der beiden CAN-Controller eine priorisierte Datenbasis zuzuordnen. Für Botschaften, die von diesem Controller empfangen werden, hat bei der Symbolzuordnung diese Datenbasis dann die höchste Priorität. Erst wenn dort kein symbolischer Namen gefunden wird, werden alle weiteren im Datenbasisauswahldialog angegeben Datenbasen in der dort festgelegten Reihenfolge durchsucht.

Zur Auflösung von Namenskonflikten bei der Konfiguration von Meßfenstern und Funktionsblöcken wird ebenfalls die Suchreihenfolge in der Datenbasisliste im Datenbasisauswahldialog herangezogen. Dem Namen wird in diesem Fall die Botschaft der am weitesten oben stehenden Datenbasis zugeordnet. Sie haben jedoch die Möglichkeit Mehrdeutigkeiten dieses Typs aufzulösen, indem Sie symbolische Bezeichner qualifizieren.

Beispiele zur Auflösung von Mehrdeutigkeiten finden Sie in der CANalyzer Online-Hilfe.

4.3.4 Überprüfung der Konsistenz symbolischer Daten

Wenn Sie während Ihrer Arbeit mit dem CANalyzer zugeordnete Datenbasen modifizieren, können Inkonsistenzen auftreten. Um dies zu vermeiden, überprüft der CANalyzer in folgenden Situationen die Konsistenz der Datenbasen mit der aktuellen Konfiguration

- bei Programmstart,
- bei Zuordnen einer neuen Datenbasis
- bei erneutem Aktivieren des CANalyzers nach Änderung der Datenbasis.

Bei der Konsistenzüberprüfung werden die symbolischen Namen aller im Meßaufbau verwendeten CAN-Botschaften mit den Namen aus den Datenbasen verglichen. Hat sich der Botschaftsname, nicht aber der Identifier, in der Datenbasis geändert, so wird der Name automatisch angepaßt. Dabei erscheint eine entsprechende Mitteilung am Bildschirm. Falls der CANalyzer weder den Namen noch den Botschafts-Identifier in den Datenbasen findet, erhalten Sie eine Fehlermeldung. Sie können die Messung in diesem Fall erst starten, wenn Sie die entsprechende Botschaft aus der Konfiguration entfernt haben.

4.4 Die DDE-Schnittstelle

Die DDE-Schnittstelle (Dynamic Data Exchange) bietet Windows-Programmen die Möglichkeit, Daten mit dem CANalyzer auszutauschen. Ein als DDE-Client bezeichnetes Windows-Programm (z.B. Excel, Word oder der Explorer) baut dabei die Verbindung zum CANalyzer - dem DDE-Server - auf, fordert Daten an bzw. setzt Kommandos zum CANalyzer ab.

Alle DDE-Dienste der CANalyzer anbietet, sind unter dem Servicenamen `CANalyzerDDEMLServer` zusammengefaßt. Diesen Namen müssen Sie Ihrer Client-Anwendung mitteilen. Mit dem Themennamen („topic name“) werden schließlich für jeden Service bestimmte Dienste zusammengefaßt. Der CANalyzer stellt unter dem Thema `System` verschiedene Dienste bereit. Als Datenformat wird dabei ausschließlich CF_TEXT (ASCII-Text) unterstützt.

Wie Sie vorhandene Windows-Programme zur Datenkommunikation über DDE einsetzen, finden Sie in der Dokumentation dieser Programme beschrieben. Sie können natürlich auch eigene Windows-Anwendung schreiben (z.B. in Visual Basic, Visual C oder Borland C), die als DDE-Client mit dem CANalyzer kommunizieren. Als typische DDE-Anwendungen wird das Laden von Konfigurationen mit dem Explorer im Abschnitt 4.4.4 beschrieben.

4.4.1 DDE-Dienste

Die DDE-Schnittstelle faßt Datenanforderungen und Kommandoaktionen unter dem Begriff *Transaktion* zusammen. Man unterscheidet 4 verschiedenen Klassen von Transaktionen:

Datenanfrage (Peek)

Mit dieser Transaktion können gezielt Daten beim DDE-Server gelesen werden. So läßt sich z.B. Abfragen, ob die Messung gerade läuft oder in welchem Modus sich der CANalyzer befindet.

Datenmanipulation (Poke)

Mit dieser Transaktion können Daten beim DDE-Server manipuliert werden.

Aufsetzen einer Benachrichtigungsschleife (Advise Loop)

Die Benachrichtigungsschleife bietet eine Möglichkeit, den DDE-Client gezielt über Werteänderungen bestimmter Daten zu informieren.

Ausführen von Kommandos (Execute)

DDE-Clients können gezielt Kommandos an den DDE-Server absetzten und ihn damit fernsteuern. Beispielsweise kann ein DDE-Client den CANalyzer veranlassen, eine bestimmte Konfiguration zu laden, die Messung zu starten oder in den Offline-Modus zu wechseln.

Der CANalyzer kann ohne Einschränkungen eine beliebige Anzahl von Transaktionen gleichzeitig führen.

Zur Spezifizierung der bei Transaktionen übertragenen Daten verwendet DDE eine dreistufige Hierarchie: Ganz oben steht dabei der *Servicename*, mit der ein Client den Server beim Verbindungsaufbau anspricht. Wie die meisten DDE Server stellt der CANalyzer genau einen Servicenamen – die Zeichenkette `CANoeDDEMLServer` – bereit. Mit dem *Themennamen* werden auf der darunterliegenden Ebene für jeden Service logische Dateneinheiten zusammengefaßt. Als Datenformat unterstützt der CANalyzer dafür ausschließlich `CF_TEXT` (ASCII-Text). Auf der untersten Ebene steht schließlich das *Item*, das eine Dateneinheit eindeutig beschreibt. Diese gibt der Server im Verlauf der Transaktion an den Client weiter. Ein Item kann dabei ein Integer, eine Zeichenkette oder auch eine Bitmap sein. Der CANalyzer stellt Ihnen unter dem Thema *System* die beiden Items *ConfigurationRequest* und *DynamicStateRequest* zur Verfügung, mit denen Sie Informationen über den aktuellen Systemzustand abfragen können.

4.4.2 Das DDE Thema *System*

Mit dem Thema *System* können Sie sowohl Systeminformationen beim CANalyzer abfragen als auch Kommandos an den CANalyzer absetzen.

Zur Abfrage von Systeminformationen stellt der CANalyzer einen DDE-Kanal bereit, mit dem ein DDE-Client Statusmeldungen lesen kann (Peek) und über Statusänderungen informiert wird (Advise Loop).

Die Items *ConfigurationRequest* und *DynamicStateRequest* übermitteln dabei Daten über den aktuellen Zustand der Konfiguration. Der Zustand kann dabei über eine Advise Loop überwacht oder durch einen Peek auf das betreffende Item erfragt werden.

Die in den beiden folgenden Tabellen dokumentierten Rückgabewerte der Items *ConfigurationRequest* und *DynamicStateRequest* sind mit `oder` verknüpft. Der Rückgabewert 6 für des Item ConfigurationRequest bedeutet also, daß die Konfiguration modifiziert wurde und der Online-Modus aktiv ist.

Rückgabewerte von *ConfigurationRequest*:

Wert	Bedeutung
1	Konfiguration hat einen Namen
2	Konfiguration ist modifiziert/noch nicht gespeichert
4	Modus ist Online

Rückgabewerte von *DynamicStateRequest*

Wert	Bedeutung
0	Messung läuft nicht
4	Messung läuft

Eine Datenanforderung auf das Item *Formats* liefert die unterstützten Datenformate. Dies ist momentan nur `TEXT`. Dies entspricht dem nach der DDEML-Konvention benannten Zwischenablageformat `CF_TEXT`.

Mit der DDE Transaktion *Execute* wird die Ausführung eines CANalyzer-Befehls über DDE veranlaßt. Es existiert zur Zeit noch keine echte Rückgabe über Erfolg oder das Ergebnis des Befehles. Einige Befehlsergebnisse lassen sich jedoch über die Items *ConfigurationRequest* und *DynamicStateRequest* erfragen.

4.4.3 Die DDE Transaktion *Execute*

Die zur Fernsteuerung des CANalyzers verwendeten Kommandos bestehen aus einem null-terminierten Kommandostring.

Dieser Kommando-String besteht aus einem Kommando-Token und - falls erforderlich - aus Kommandoparametern, die durch Kommata getrennt werden und optional durch runde Klammern eingeklammert werden dürfen. String-Parameter dürfen nicht in Anführungszeichen gesetzt werden. Der Kommando-String darf in eckige Klammern gesetzt werden (DDEML-Konvention). Eine einzelne *Execute*-Transaktion kann nur genau ein Kommando absetzen, d.h. Skripte oder Folgen von Kommandos sind nicht zulässig. Die Kommando-Tokens unterscheiden nicht zwischen Groß- und Kleinschreibung und werden von Leerzeichen begrenzt.

Folgende Kommandos werden unterstützt:

Kommando	Bedeutung	Bemerkungen
Start	startet die Messung	wird während laufender Messung ignoriert
Stop	stoppt die Messung	wird außerhalb der laufenden Messung ignoriert
ToOffline	schaltet in Offline-Modus	wird während der Messung oder im Offline-Modus ignoriert
ToOfflineCopy	schaltet in Offline-Modus (Kopie)	wird während der Messung oder im Offline-Modus ignoriert
ToOnline	schaltet in Online-Modus	wird während der Messung oder im Online-Modus ignoriert
ToOnlineCopy	schaltet in Online-Modus (Kopie)	wird während der Messung oder im Online-Modus ignoriert
LoadCfg <name>	lädt die Konfiguration <name>	Falls <name> leer ist, wird der Dateiauswahldialog geöffnet.
SaveCfg <name>	sichert die Konfiguration unter dem Namen <name>	Falls <name> leer ist, wird der Dateiauswahldialog geöffnet.

Beispiele für *Execute*

Die folgenden Zeilen zeigen einige Beispiele für Kommandostrings, die mit der Transaktion *Execute* abgesetzt werden können:

```
LoadCfg c:\canwin\demo_cn\easy\easy.cfg
LoadCfg (c:\canwin\demo_cn\easy\easy.cfg)
[LoadCfg (c:\canwin\demo_cn\easy\easy.cfg)]
LoadCfg
Start
Stop
```

4.4.4 Laden von Konfigurationen mit dem Explorer

Um CANalyzer-Konfigurationen (CFG-Dateien) direkt über den Explorer zu laden, genügt es nicht, die Erweiterung `CFG` im Explorer mit der Applikation `CANW.EXE` bzw. `CANW32.EXE` zu verknüpfen. Ist der CANalyzer nämlich bereits gestartet, erhalten Sie bei dem Versuch, durch Doppelklicken auf eine CFG-Datei eine neue Konfiguration zu laden, eine Fehlermeldung, denn der Explorer versucht in diesem Fall nach dem Doppelklicken den CANalyzer ein zweites Mal zu starten.

Mit Hilfe der DDE-Schnittstelle von läßt sich der Explorer jedoch auch so konfigurieren, daß bei bereits gestartetem Programm die gewünschte Konfiguration geladen wird. Gehen Sie dazu folgendermaßen vor:

Löschen der Zuordnung

Lösen Sie im Explorer zunächst eine evtl. bereits bestehenden Zuordnung zu CFG-Dateien. Doppelklicken Sie zunächst auf eine CFG-Datei. Wenn daraufhin ein Programm startet, so besteht bereits eine Zuordnung zu CFG-Dateien. Erscheint hingegen beim Doppelklicken der Dialog *Öffnen mit...*, dann besteht noch keine Zuordnung und Sie können eine neue Zuordnung direkt erstellen.

Eine Zuordnung können Sie sicher löschen, indem Sie sie in der Registry löschen. (Es gibt verschiedene Arten von Zuordnungen, manche lassen sich evtl. auch einfacher - direkt im Explorer - lösen.) Um die Zuordnung in der Registry zu löschen, öffnen Sie zunächst den Registrierungseditor z.B. durch *Start|Ausführen regedit <enter>*.

Suchen Sie dann den Schlüssel `HKEY_CLASSES_ROOT\.cfg` und löschen Sie diesen im lokalen Kontextmenü zu diesem Schlüssel.

Erstellen der neuen Zuordnung

Eine neue Zuordnung erstellen Sie im Explorer mit dem Menüpunkt *Ansicht|Optionen*. Wählen Sie in der Registerkarte *Dateitypen* die Schaltfläche *Neuer Typ*. Es erscheint der Dialog *Neuen Dateityp hinzufügen*. Drücken Sie hier zunächst die Schaltfläche *Anderes Symbol*. Suchen Sie hier nach der CANalyzer-Applikation `EXEC\CANW.EXE` (16-bit) bzw. `EXEC32\CANW32.EXE` (32-bit), wählen Sie das CANalyzer-Symbol und bestätigen Sie mit *OK*.

Tragen Sie im Textfeld *Beschreibung des Dialogs* die Beschreibung ein, die im Explorer angezeigt werden soll, z.B. „CANalyzer Konfiguration“. Die zugeordnete Erweiterung muß hier natürlich `CFG` lauten. Lassen Sie das Textfeld *Inhaltstyp(MIME)* leer. Das Textfeld *Standarderweiterung für Inhaltstyp* ist wahrscheinlich deaktiviert. Lassen Sie es ebenfalls leer.

Fügen Sie dann einen neuen Vorgang ein, indem Sie die Schaltfläche Vorgänge drücken. Es erscheint der Dialog *Neuer Vorgang*. Tragen Sie hier als Vorgang ein *open*. Als *Anwendung für diesen Vorgang* wählen Sie mittels Durchsuchen die CANalyzer-Applikation `EXEC\CANW.EXE` (16-bit) bzw. `EXEC32\CANW32.EXE` (32-bit). Kreuzen Sie anschließend DDE verwenden an, und tragen Sie in dem nunmehr aufgeklappten Teil des Dialogs die DDE-Informationen so ein, wie in Abbildung 68 beschrieben.

Nun können Sie die Konfigurationsdateien mit der Erweiterung `CFG` doppelklicken. Dies funktioniert im Explorer und in Dateigruppen. Ist der CANalyzer noch nicht gestartet, wird es hochstarten und nach dem Hochstarten die gewünschte Konfiguration geladen (Blinkeffekt). Ist der CANalyzer schon geladen, wird die Konfiguration gewechselt. Dies funktioniert auch mit Verknüpfungen. Sie können also Verknüpfungen von Konfigurationsdateien auf dem Desktop und in Programmgruppen anlegen, die sich dann auf Doppelklick verhalten, wie die Datei selber.

Abbildung 68: Erstellen der DDE-Anbindung für den Explorer

Hinweis: Da der CANalyzer und CANoe die gleiche Extension für Ihre Konfigurationen verwenden, läßt sich auf einem System nur eines der beiden Programme so konfigurieren, wie hier beschrieben. Zum Öffnen des zweiten Programmes können jedoch einen weiteren Vorgang (z.B. *open CANoe*) für den Dateityp CFG hinzufügen.

5 Anhang

5.1 Besonderheiten der Demo-Version

Bei der Demo-Version des CANalyzers wird anstelle eines regulären PC-Kartentreibers ein Demo-Treiber dazugebunden, der keine PC-Karte benötigt. Die Funktionen dieses Treibers sind allerdings sehr eingeschränkt. Er sorgt hauptsächlich dafür, daß alle Botschaften, die gesendet werden, als empfangene Botschaften mit dem aktuellen Zeitstempel zurückgegeben werden.

Um mit der Demo-Version arbeiten zu können, muß deshalb im Sendezweig ein Generator- oder Programmblock eingefügt werden, der Sendebotschaften erzeugt. Die so generierten Botschaften können dann erfaßt, ausgewertet und abgespeichert werden.

Die Einstellungen der Busparameter und das Botschafts-Setup, die man über das Anklicken der PC-Karte im Datenflußbild erreicht, sind für die Demo-Version belanglos und können vernachlässigt werden.

Die Demo-Version ist abgesehen von der PC-Karte und dem zugehörigen Kartentreiber eine vollwertige Version. Insbesondere kann die Auswertung und Abspeicherung von Botschaften und die CAPL-Programmierung ohne Einschränkungen getestet werden.

5.2 Verwendete Dateinamen

Zusätzlich zu den Dateien, die bei der Installation angelegt werden, verwendet das Programm eine Reihe weiterer Dateien, die z.B. Meßaufbauten oder Logging-Daten speichern. Zur Unterscheidung der Dateitypen werden verschiedene Extensionen verwendet:

Extension	Dateiart
AC	Firmware zum Download in CAN-AC2 Karten
ASC	ASCII-Logging-Datei, Replay-Datei
BAK	Backup für CAPL-Quelltexte
CAN	CAPL-Quelltext
CBF	CAPL-Programmdatei (compiliert)
CFG	Konfigurationsdatei für Meßaufbau
DBC	Projektdatenbasis (ASCII-Format)
DLL	Laufzeitbibliotheken
EXE	Programmdateien
GEN	gespeicherter Generatorblock
HLP	Hilfedateien
HEX	Intel-Hex-Datei als Firmware zum Download
INI	Windows Initialisierungsdateien
LOG	Binäre Logging-Datei, Replay-Datei
SBN	Firmware zum Download bei PCMCIA CANcard
TXT	ASCII-Dokumente mit weiteren Informationen
WMF	Ressourcen im Windows Metafile-Format
WRI	Write-Dokumente mit weiteren Informationen

5.3 Trouble shooting

Eine Liste der während der Initialisierung oder einer Messung auftretenden Fehlermeldungen findet sich in Abschnitt 5.4. Hier werden Tips zur Behandlung von Problemen allgemeinerer Art gegeben.

Der CANalyzer startet nicht

CFG-Datei zerstört? Oft hilft es, die aktuelle Konfigurationsdatei XYZ.CFG zu löschen. Um den Inhalt nicht zu verlieren, sollte die Datei unter einem anderen Namen zwischengespeichert werden. Nach Klärung des Problems kann sie wieder in XYZ.CFG umbenannt werden.

Der CANalyzer läuft zu langsam

Für den Betrieb des CANalyzers darf kein Power-Manager installiert sein, wie er vor allem für Notebooks üblich ist. Dieser entzieht der Anwendung unter anderem für längere Zeit die CPU. Daher stimmen bei installiertem Power-Manager die Sendezeiten nicht und es können Botschaften verloren gehen. Um den Power-Manager von Ihrem System zu entfernen, vergleichen Sie bitte auch die Hinweise zur Installation der Hardware.

Bei weniger leistungsfähigen Rechnern kann es darüber hinaus sinnvoll sein, die Auflösung des Systemzeitgebers zu verringern. Die Zeitstempel der Botschaften sind dann u.U. weniger genau, aber die Rechner-CPU wird weniger belastet. Tragen Sie dazu in der Datei `CAN.INI` im Abschnitt [PCBOARD] die Zeile

```
Timerrate = 200
```

bzw. u.U. sogar den Wert

```
Timerrate = 400
```

ein. Dies entspricht einer Zeitauflösung von 2 bzw. von 4 Millisekunden.

Karte kann nicht initialisiert werden

Timeout ...

Bei Fehlermeldungen dieser Art kann der CANalyzer keine Verbindung zur CAN-Hardware aufnehmen. Überprüfen Sie die Installation der CAN-Karte und die Dokumentation der Hardware. Hinweise zur Fehlerbehebung finden Sie im Anhang der Hardware-Installationsanleitung.

Vor allem Notebooks verwenden häufig einen Power-Manager. Dieser *muß* deaktiviert werden! (Vgl. Hinweise zur Installation der Hardware.)

Fehlermeldung "Fehler beim Senden"

Sofortige Zustandsänderung der CAN-Controller in ERROR PASSIVE

Bus nicht angeschlossen? Der Busanschluß ist zu überprüfen, evtl. auch die Pinbelegung des verwendeten Steckers.

Abschlußwiderstand vorhanden? Insbesondere die CAN AC2 Version mit 82527 Controllern reagiert empfindlich auf das Fehlen eines Busabschlusses.

Kein Partner am Bus? Wird der Bus nur an einen der beiden CAN-Controller angeschlossen und existiert kein weiterer Busteilnehmer, so erhält der Controller bei Sendeversuchen kein Acknowledge.

Baudrate und Output Control eingestellt? Die Programmierung der Controller-Register kann über das Kontextmenü der CAN-Karten-Sinnbilder erreicht werden. Eine genaue Beschreibung finden Sie in Abschnitt 2.2.

Meldung:
Ein Fehler ist aufgetreten in ...
Für weitere Informationen melden Sie bitte
ZEILE (line) und DATEI (file) an unsere Hotline

Beim derzeitigen Stand der Technik ist es nicht möglich, vollkommen fehlerfreie nichttriviale Programme zu entwickeln. Dies gilt leider auch für den CANalyzer. Um seltene Systemfehler besser lokalisieren und beheben zu können, verfügt der CANalyzer über Diagnosemechanismen, die diesen Fehler melden. Bitte notieren Sie sich den genauen Text der Fehlermeldung. Unser telefonischer Kundendienst kann Ihnen mit diesen Angaben schneller helfen, das Problem zu beseitigen.

5.4 Meldungen im Write-Fenster

Im folgenden finden Sie Beschreibung der wichtigsten CANalyzer-Systemmeldungen, die ins Write-Fenster ausgegeben werden.

`Assertion <Fehlernummer> in <Dateiname> <Bedingung>`

Beim derzeitigen Stand der Technik ist es nicht möglich, vollkommen fehlerfreie nichttriviale Programme zu entwickeln. Dies gilt leider auch für den CANalyzer. Um seltene Systemfehler besser lokalisieren und beheben zu können, verfügt der CANalyzer über Diagnosemechanismen, die diesen Fehler melden. Bitte notieren Sie sich den genauen Text der Fehlermeldung. Unser telefonischer Kundendienst kann Ihnen mit diesen Angaben schneller helfen, das Problem zu beseitigen.

`board error: <Fehlernummer>`

Ein Fehler an der Schnittstelle zur CAN-Einsteckkarte ist aufgetreten. Nähere Informationen finden Sie mit Hilfe der Fehlernummer in Abschnitt 5.4.

`Bus mit <Baudrate> Baud.`

Beim Start der Messung meldet Ihnen der CANalyzer die Baudrate, auf die die PC-Einsteckkarte in der aktuellen Konfiguration initialisiert wird. (Zur Konfiguration der PC-Karte vgl. Abschnitt 2.2.)

`can board error: <Fehlernummer>`

Ein Fehler an der Schnittstelle zur CAN-Einsteckkarte ist aufgetreten. Nähere Informationen finden Sie mit Hilfe der Fehlernummer in Abschnitt 5.4.

can board: board timeout

vgl. Meldung *Timeout* in Abschnitt 5.4.

can board: dpram overflow

vgl. Meldung *DPRAM Überlauf* in Abschnitt 5.4.

can board: rx queue is full

vgl. Meldung *Rx-Puffer Überlauf* in Abschnitt 5.4.

can board: rx queue is full and will be deleted

vgl. Meldung *Rx-Puffer Überlauf* in Abschnitt 5.4.

can board: rx reg overrun

vgl. Meldung *Rx-Register Überlauf* in Abschnitt 5.4.

can board: tx queue is full

vgl. Meldung *Tx-Puffer voll* in Abschnitt 5.4.

CANcard box trigger command <Kanal> <Kommando>

Für Lowspeed-Anwendungen mit der CANcard können Sie beide Triggerkanäle der Bustreiberbox zu Messungsstart und –stopp bzw. mit der CAPL-Funktion `setPortBits()` schalten. Die Kommandos tragen Sie im Abschnitt `[CANcard]` in der Datei `CAN.INI` ein. Beim Schalten der Kanäle erscheint die entsprechende Meldung im Write-Fenster

Delayed loading of replay block

Große Replay-Dateien passen u.U. nicht in den Windows-Hauptspeicher. In diesem Fall lädt der CANalyzer während der Messung Teile der Replay-Datei neu. Die Warnmeldung erscheint, falls die Daten nicht schnell genug nachgeladen werden können. Sie müssen dann damit rechen, daß die Botschaften erst mit einer zeitlichen Verzögerung auf den Bus ausgegeben werden.

DEMO Version: Messung gestoppt !!

Diese Meldung weist Sie darauf hin, daß Sie die Messung einer Demo-Version beendet haben.

Empty replay block ignored

Eine Replay-Datei, die Sie über einen Replay-Block abspielen möchten, ist leer. Der Replay-Block wird daher ignoriert.

Error parsing file <Dateiname>: no database assigned

Wenn Sie Replay-Dateien abspielen, die Sie zuvor im symbolischen Modus mit der Logging-Funktion aufgezeichnet haben, sollten Sie darauf achten, daß beim Abspielen die gleiche Datenbasis zugeordnet ist. Bei ASCII-Dateien werden keine Botschafts-Identifier sondern die entsprechenden symbolischen Namen aus der Datenbasis gespeichert. Falls Sie zum Abspielen eine andere Datenbasis verwenden, können die symbolischen Namen beim Einlesen der Replay-Datei u.U. nicht bzw. nicht richtig interpretiert werden.

Error: unknown environment variable in <Dateiname> in line <Zeilennummer>

Die ASCII-Replay-Datei enthält eine Umgebungsvariable, die nicht in der Datenbasis gefunden wurde. Vergewissern Sie sich, daß die Datenbasis, die Sie beim Aufzeichen der Datei verwendet haben und die Datenbasis, die Sie beim Abspielen der Datei verwenden, übereinstimmen.

Fehler: Nicht genug Speicher zum Erzeugen des Replay-Blocks!

Beim Start der Messung wird die Replay-Datei, die Sie dem Replay-Block zugeordnet haben zunächst in den Hauptspeicher eingelesen. Die Meldung erscheint, wenn Windows nicht mehr genug Hauptspeicher bereitstellen kann, um den Replay-Block zu laden. Schließen Sie einige Anwendungen, bevor Sie die Messung erneut starten.

Fehler: Nicht genug Speicher zum Laden der Offline-Source!

Beim Start der Messung im Offline-Modus wird die Offline-Datenquelle zunächst aus der Offline-Datei in den Hauptspeicher eingelesen. Die Meldung erscheint, wenn Windows nicht mehr genug Hauptspeicher bereitstellen kann, um die Offline-Datenquelle zu laden. Schließen Sie einige Anwendungen, bevor Sie die Messung erneut starten.

Kein Windows-Timer für Animation verfügbar

Für den Animationsbetrieb im Offline-Modus wird ein Timer des Windows-Betriebssystems verwendet. Ist kein Timer mehr verfügbar, erscheint diese Meldung. Die Messung wird danach abgebrochen. Schließen Sie einige Anwendungen, die Windows-Timer verwenden, bevor Sie die Messung erneut im Animationsbetrieb starten.

Lastübergang:

Der Ringpuffer zwischen Echtzeitbibliothek und dem CANalyzer-Hauptprogramm verfügt über eine eigene Füllstandsanzeige. Falls der Puffer aufgrund hoher Systembelastung voll- bzw. überläuft, erscheint eine entsprechende Meldung (`NORMAL->QUEUE HIGH` bzw. `NORMAL->DATA LOST`) im Write-Fenster. Geht die Belastung (etwa nach einem nach einem Botschaftsburst) wieder zurück, werden Sie auch darüber durch die Meldung `QUEUE HIGH -> NORMAL` informiert.

No interrupts from CAN PC board received !!
System has problems. Check CAN board or CAN.INI for IRQ configuration. IRQ currently = <IRQ>

Die Kommunikation zwischen der PC-Einsteckkarte und dem CANalyzer wird normalerweise über einen Interrupt gesteuert. Diese Fehlermeldung weist darauf hin, daß der CANalyzer keine Interrupts von der Karte empfangen hat. Vergewissern Sie sich, daß Sie den Interruptkanal für die Einsteckkarte auch in der Datei `CAN.INI` dem CANalyzer bekannt gemacht. Das genaue Vorgehen ist in der Installationsanleitung beschrieben. Die in der Fehlermeldung angegebene IRQ-Nummer entspricht dem aktuellen Interruptkanal, der in Ihrer INI-Datei eingestellt ist.

PB1/2 initialized by CAN.INI : <Initialisierungswert>
PB3 initialized by CAN.INI: <Initialisierungswert>
PB1/2 initialized by USRDLL: <Initialisierungswert>
PB3 initialized by USRDLL: <Initialisierungswert>

Für Lowspeed-Anwendungen können die Lowspeed-Treiber („Piggy-Backs“) auf der CAN-AC2 Karte über die Datei `CAN.INI` , mit den CAPL-Funktionen `setPortBits()` und `enablePortBits()` oder über eine zur Konfiguration gehörende Laufzeitbibliothek initialisiert werden. Die Initialisierungswerte aus der INI-Datei oder aus der Laufzeitbibliothek werden im Write-Fenster ausgegeben. Hinweise zur Konfiguration der Lowspeed-Treiber finden Sie in der Installationsanleitung für den Lowspeed-Umrüstsatz.

PC-Einsteckkarte nicht gefunden. Fehlernummer <Fehlernummer>

Der CANalyzer hat beim Programmstart die PC-Einsteckkarte nicht erkannt. Die Fehlernummer finden Sie in der Liste der Fehlermeldungen zur CAN-Schnittstelle (Abschnitt 5.4). Vergewissern Sie sich, ob Sie die Karte wie in der Installationsanleitung beschrieben installiert haben und ob die Einstellungen in der Datei `CAN.INI` mit den Hardwareeinstellungen (Speicherbereich, Interruptkanal) übereinstimmen.

Replay: Laden wurde verzögert

Große Replay-Dateien passen u.U. nicht in den Windows-Hauptspeicher. In diesem Fall lädt der CANalyzer während der Messung Teile der Replay-Datei neu. Die Warnmeldung erscheint, falls die Daten nicht schnell genug nachgeladen werden können. Sie müssen dann damit rechen, daß die Botschaften erst mit einer zeitlichen Verzögerung auf den Bus ausgegeben werden.

RTC has to be activated for polling mode (CAN.INI)

Sie haben den CANalyzer in der Datei `CAN.INI` vom interruptgesteuerten Standardmodus in den Pollingmodus umgeschaltet. Sie müssen in diesem Modus zusätzlich die PC-Echtzeituhr (RTC) als Zeitbasis aktivieren. Stellen Sie

dazu im Abschnitt [RTC] der INI-Datei den Eintrag Frequency auf einen Wert zwischen 8 und 11 ein.

Beachten Sie, daß der CANalyzer standardmäßig im Interrupt getriebenen Modus arbeitet. Verwenden Sie den Pollingmodus lediglich in Ausnahmesituationen, um bei sehr hoher Grundlast den Datendurchsatz geringfügig zu erhöhen bzw. wenn Sie nicht mit Interrupts arbeiten können oder wollen.

RTC is activated (see CAN.INI section RTC)

Sie haben in der Datei CAN.INI als Zeitgeber die PC-Echtzeituhr (RTC) gewählt (Abschnitt [RTC], Eintrag Frequency ist ungleich 0). Normalerweise wird die Zeitverwaltung vom Zeitgeber auf der PC-Einsteckkarte gesteuert (Frequency = 0). Die PC-Echtzeituhr sollte nur in Ausnahmesituationen (z.B. im Pollingmodus) aktiviert werden.

SetTimer warning: timer still active. (No is the timer corresponding to the n-th [0.,1.,..] ON TIMER in text file of CAPL program)

Sie verwenden in einem CAPL-Programm einen Timer, der mit der Funktion setTimer() aufgezogen wird, noch bevor er nach dem letzten Aufziehen abgelaufen ist. In einem CAPL-Programm mit mehreren Timern können Sie diesen lokalisieren, indem Sie das CAPL-Programm mit einem Texteditor öffnen und die on timer-Ereignisprozeduren – beginnend mit 0 – von oben nach unten durchzählen. Die in der Warnung angegebene Timer-Nummer entspricht der Prozedur-Nummer der so durchgezählten Ereignisprozeduren.

Timer Warning: no completion function, SetTimer ignored

Sie versuchen in einem CAPL-Programm mit der Funktion setTimer() einen Timer aufzuziehen, für den Sie keine on timer-Ereignisprozedur definiert haben. Der Funktionsaufruf wird daher ignoriert.

Warning: can't set timer; no completion function

Sie versuchen in einem CAPL-Programm mit der Funktion setTimer() einen Timer aufzuziehen, für den Sie keine on timer-Ereignisprozedur definiert haben. Der Funktionsaufruf wird daher ignoriert.

Warnung: Leerer Replay-Block ignoriert

Eine Replay-Datei, die Sie über einen Replay-Block abspielen möchten, ist leer. Der Replay-Block wird daher ignoriert.

5.5 Liste der Fehlermeldungen zur CAN-Schnittstelle

In dieser Liste erscheinen die Fehlermeldungen, die sich auf die Kommunikation zwischen dem CANalyzer und der CAN-PC-Karte beziehen sowie Meldungen von Fehlern auf dem CAN-Bus oder in der CAN-Karten-Firmware. Es ist jeweils der

Klartext und eine zugeordnete Fehlernummer angegeben. Einige dieser Meldungen sind hardware-spezifisch, treten also nicht bei allen Kartentypen auf.

Botschaft wurde nicht gesendet (14)

Die letzte Sendeanforderung wurde vom CAN-Controller nicht durchgeführt. Dies kann sowohl am Fehlerstatus des Controllers liegen, als auch an einem Aufkommen zu vieler höherpriorer Botschaften.

DLC-Fehler (1128, 111, 112)

Bei der CANIB-Karte ist im DPRAM-Setup ein kleinerer DLC angegeben, als gesendet werden soll. Im CAPL-, Generator- bzw. Replay-Block ist diese Größe anzupassen.

Der Fehler kann Folgefehler auslösen. Bei Problemen hierbei ist der Rechner mit einem Kaltstart neu zu booten!

Aufgrund eines Firmware-Fehlers der CANIB kann ein DLC gleich 0 nicht verarbeitet werden. Er wird daher im Treiber auf mindestens 1 gesetzt. Ein Ausgleich bei der Berechnung der freien DPRAM-Größe erfolgt aber erst ab Version 2.0.

DPRAM-Überlauf (5)

Es wurden im Botschafts-Setup mehr Botschaften definiert, als das DPRAM aufnehmen kann. Da der CANalyzer die Anzahl der eintragbaren Botschaften prüft, kann diese Meldung nicht erscheinen.

Falsche Controller-Nr. (3,10,113)

Es wird versucht, auf einen nicht vorhandenen CAN-Controller zuzugreifen. Die meisten vom CANalyzer unterstützten CAN-Karten haben zwei Controller. Es gibt aber auch Karten mit nur einem.

Abhilfe: In den CAPL-Programmen nach einem "message CANx... " suchen, wobei x höher als die Zahl der vorhandenen Controller ist. Dieses durch eine korrekte Zahl ersetzen.

Falsche Modulnummer (109)

An der angegebenen Adresse ist keine CANIB gefunden worden. Ist auf der Karte die Port-Adresse umgejumpert worden, so muß dies dem CANalyzer mitgeteilt werden. Siehe Handbuch.

Falsche Prüfsumme (1368)

Der CAN-Controller hat eine fehlerhafte CRC-Prüfsumme festgestellt.

Falscher Abschlußcode (1432)

Bei CANIB-Karten Fehlercode für allgemeine Probleme der Firmware. Taucht insbesondere im Status BUSOFF und ERROR_PASSIVE auf.

Falscher Kartentyp (8)

Der Kartentreiber des CANalyzers und die Hardware passen nicht zusammen.

Abhilfe: Starten Sie die richtige CANalyzer-Version oder bauen Sie eine geeignete CAN-Karte ein.

Fehler bei Statistikanforderung ()

Die CAN-Karte oder deren Firmware unterstützt keine Busstatistik.

Fehlerhafte ID, DLC usw. (1544)

Kann bei CANalyzer Windows nicht auftreten.

Ausnahme: CANIB. Beschreibung siehe 1128.

FIFO-Eintrag>16 (114)

Die CANIB hat ungültige Daten zum CANalyzer gemeldet. Eine mögliche Ursache hierfür ist ein Rx-Pufferüberlauf. Dieser kann nicht direkt erkannt werden, wenn die CANIB nicht korrekt initialisiert wurde. Sind deren Jumper umgestellt, so ist nach CANalyzer-Handbuch zu verfahren.

Interrupt nicht gefunden (108,2016)

Es konnte keine korrekte Kommunikation mit der CANIB-Karte hergestellt werden. Es ist zu prüfen, welcher Interrupt auf der CANIB-Karte gejumpert ist und ob es Kollisionen mit anderer Hardware geben kann. Ein häufiger Fall ist eine Überschneidung mit dem Mausinterrupt. Der CANIB-Treiber kann selbst den von der CANIB belegten Interrupt feststellen. Sollte dies fehlschlagen kann ab Version 2.0 die IRQ - Nummer explizit angegeben werden (siehe BOARDCFG.INI).

Kein Abschlußcode empfangen (1384)

Kann bei CANalyzer Windows nicht auftreten.

Kein Zugriff auf IMP (12)

Die Firmware hat keinen Zugriff auf den Interface Management Prozessor des Full-CAN-Chips 82526 erhalten .

Keine Antwort vom CAN-Controller (106)

Die Firmware konnte keine Verbindung zum CAN-Controller herstellen. Dies ist ein Hinweis auf eine defekte CAN-Karte.

Keine Botschaften im Rx-Puffer (1)

Keine Botschaft erhalten (7)

Es werden momentan keine Daten von der Karte empfangen.

Kommando vom Treiber nicht unterstützt (6,11,1528)

Der CANalyzer hat ein Kommando an den Kartentreiber gesendet, das dieser oder die Firmware nicht kennt. Beispiel: Eine Busstatistikanforderung auf einer Karte ohne eine entsprechende Logik.

Rx-Puffer-Überlauf (101)

Der Empfangspuffer konnte die empfangenen Botschaften nicht mehr aufnehmen.

Zur Abhilfe gibt es mehrere Möglichkeiten:

- Nicht benötigte Zweige im Datenflußdiagramm unterbrechen. Im Extremfall können der Statistikblock, der Datenblock und der Trace-Block unterbrochen, die Messung mit dem Logging-Block aufgezeichnet und anschließend offline ausgewertet werden.
- Bei Basic-CAN-Controllern kann der Datenstrom über die Akzeptanzfilterung reduziert werden. Außer bei Spezialanwendungen kann insbesondere der zweite Controller hierdurch vollkommen abgeschaltet werden.
- Bei Full-CAN-Controllern kann eine Datenreduzierung durch Streichen von Botschaften im Botschafts-Setup in Verbindung mit Filterblöcken erreicht werden.
- Abschalten der Option Statistikprotokoll oder anderer nicht benötigter Optionen.

Rx-Register-Überlauf (105)

Der Basic-CAN-Controller 82C200 besitzt nur zwei interne Register zur Aufnahme von Botschaften. Bei hoher Busfrequenz und hoher Botschaftsrate überschreiben neu ankommende Botschaften diese Puffer bevor die Firmware die Register auslesen kann.

Abhilfe: Akzeptanzfilterung einsetzen.

Timeout (1080)

Timeout bei Kartenzugriff (4,232)

Während einer Messung kommt es zu Kommunikationsproblemen mit der Firmware.

Abhilfe: Messung abbrechen und neu starten. Hilft dies nicht, kann bei manchen Karten der Reset-Knopf gedrückt werden, andernfalls ist der PC neu zu booten.

Timeout während Karteninitialisierung (0)

Keine Antwort von der Karte (1400)

Beim Initialisieren der CAN-Karte konnte keine Verbindung zur Firmware hergestellt werden.

Tx-Puffer voll, Tx-Request zurückgewiesen (2)

Der Sendepuffer ist noch voll. Die neue Sendeanforderung kann nicht bearbeitet werden. Hierfür gibt es drei mögliche Gründe:

- Der CANalyzer sendet schneller Daten, als die Firmware sie empfangen und an den CAN-Controller weiterleiten kann. Dies tritt zum Beispiel dann auf, wenn auf dem CAN-Bus höherpriore Botschaften gesendet werden.
- Die Anzahl der direkt hintereinander in einem CAPL-Programm gesendeten Botschaften ist größer als der Sendepuffer. Dieses Problem gibt es vor allem, wenn in CAPL-Programmen in einer Schleife gesendet wird:

  for (i=0;i<50;i=i+1) output(Msg);

  Abhilfe: Schnelles Senden durch Setzen von msTimern und Reaktion auf den Timer-Event.
- Der angesprochene CAN-Controller ist im BUSOFF-Zustand und kann somit keine Sendeaufträge mehr entgegennehmen. Dies ist im Busstatistikfenster zu erkennen.

Übertragungsfehler (1352)

Kann bei CANalyzer Windows nicht auftreten.

Unbekannte Sende-Identifier (100,1496)

Es soll eine Botschaft gesendet werden, die nicht im Botschafts-Setup des Full-CAN-Controllers eingetragen ist.

Ungültige DPRAM-Adresse (107)

Bei der Initialisierung der CAN-Karte ist eine ungültige Kennung für die Adresse des DPRAM angegeben worden. Siehe Karten-Dokumentation.

5.6 Zusatzprogramme

5.6.1 ASC2MDF.EXE

Dieses Programm dient zur Umwandlung von ASCII-Dateien in Binärdateien im MDF-Format, die dann z.B. als Replay-Dateien verwendet werden. Im Normalfall werden für Replay-Dateien zuvor aufgezeichnete Logging-Dateien verwendet. In manchen Anwendungen soll aber ein Replay 'von Hand' generiert werden. Dazu ist mit einem Editor (z.B. EDIT.EXE von MS-DOS) eine ASCII-Datei zu erzeugen, die mit ASC2MDF.EXE in eine Binärdatei konvertiert wird.

Format der ASCII-Datei

Die Datei enthält als erste Zeile die Angabe der Zahlenbasis:

"base hex" oder "base dec"

Darauf folgend werden die zu sendenden Botschaften in jeweils einer Zeile eingetragen. Das Zeilenformat entspricht vollständig dem des Trace-Fensters.

Zur einfachen Erstellung erster Testdateien wird folgendes Vorgehen empfohlen:

Logging-Block aktivieren, Dateiformat ASCII auswählen, Dateiname TEST.ASC auswählen und eine Messung durchführen. Die erzeugte ASCII-Logging-Datei hat das erforderliche Format. Alle Zeilen, die keine Botschaften enthalten werden ignoriert. Datei nach Bedarf ändern. Dabei können Kommentarzeilen durch Voranstellen eines Semikolons erzeugt werden. ASC2MDF aufrufen:

```
ASC2MDF /A=TEST.ASC /M=TEST.LOG
```

Die erzeugte Datei TEST.LOG kann nun im Replay-Block als Quelle angegeben werden.

Aufruf und Parameter von ASC2MDF:

```
ASC2MDF [/A=asciiFile] [/M=mdfFile] [/F=headerFile] [/H]
```

/A Name der zu konvertierenden ASCII-Datei. Wird der ASCII-Dateiname nicht angegeben, dann wird als Voreinstellung 'a2m_data.asc' verwendet. Wird nur der ASCII-Dateiname angegeben, ist '/A=' optional.

/M Name der erzeugten Binärdatei im MDF-Format. Wird der MDF-Dateiname nicht angegeben, dann wird als Voreinstellung der Name der ASCII-Datei und die Erweiterung 'DAT' verwendet. Der CANalyzer benötigt im Offline Modus und für Replay-Blöcke Dateien im LOG-Format (Erweiterung .LOG). Es muß somit die Erweiterung LOG angegeben werden, z.B. /M=TEST.LOG .

/F Name der Fortbeschreibungsdatei. Wird der Dateiname nicht angegeben, dann wird als Voreinstellung die Datei a2m_head.txt verwendet. Die Datei a2m_head.txt enthält Beschreibungen über ihre Anwendung

/H diese Hilfebeschreibung.

Hinweis: ASC2MDF wurde für allgemeinere Zwecke entwickelt. Die Steuerung der Konvertierung erfolgt über eine Formatbeschreibungsdatei (a2m_head.txt). Für die Umwandlung in CANalyzer-Replay-Dateien darf diese Datei nicht geändert werden!

5.6.2 CANCONV.EXE

Programm zum Extrahieren einzelner Signale. Wird in Verbindung mit *CANgraph* benötigt und wird in dessen Dokumentation beschrieben.

5.6.3 LOG2TXT.EXE

Programm zur Umwandlung von Logging-Dateien in ein textuell lesbares Format.

Aufruf und Parameter von LOG2TXT:

```
LOG2TXT Datenbasis Logging-Datei Text-Datei [h]
```

Datenbasis:

Name der Datenbasis, die bei Generierung der Logging-Datei zugeordnet war.

Logging-Datei:

Logging-Datei im Binärformat, die umgewandelt werden soll.

Text-Datei:

Name der Zieldatei.

h:

Optionaler Parameter. Zeigt den Identifier und die Dateninhalte in Rohform hexadezimal an.

Die Beschreibung des Programmaufrufs erhält man auch durch Aufruf von LOG2TXT ohne Parameter.

6 Index

!

! Lastsymbol · 4-3

*

* (Überlastsymbol) · 4-3

@

@ (Überlastsymbol) · 4-3

<

<F10>-Taste · 1-20

7

72005 · 2-12

8

82527 · 2-12
82C200 · 2-12

A

Ablaufmodus · 2-48
abs() · 3-20; 3-60
Abschaltefunktion · 4-4
Absolutbetrag · 3-60
Abtastpunkt · 2-11
Abtastpunktes · 2-11
Achseneinstellungen · 2-31; 2-32
Achsenmodus · 2-34
Aktivitätsanzeige · 2-26
Akzeptanzcode · 2-12
Akzeptanzfilterung · 2-10; 2-12; 2-13; 4-5; 5-11
Akzeptanzmaske · 2-12; 2-13
Analyseblöcke · 3-68; 4-1; 4-4
Analysezweig · 4-5
Animate · 2-23; 2-60; 2-61; 2-62
Animation · 5-6
Animationsbetrieb · 5-6
Arbeitsverzeichnis · 2-8
Arbitrierung · 1-2
Array · 3-24; 3-43; 3-62; 3-66; 3-67
ASCII · 2-57; 2-58; 3-71; 5-2
Assertion · 5-4
attr · 2-20
Attribut · 3-16; 3-17; 4-6
Attributname · 3-16
Attributwert · 3-17
Aufzeichnungszeitraum · 2-55
Ausgabemodus · 2-22; 2-23; 2-31; 4-2; 4-3; 4-4
Auslösebedingung · 2-46; 2-47
Auswertefunktionen · 2-52
Auswertezweig · 3-10; 3-11; 3-12; 4-2

B

Backus-Naur-Form · 3-26
Balkendarstellung · 2-26; 2-27
Balkendiagramm · 2-24
Basic-CAN-Controller · 2-10
Baudrate · 2-11; 2-12; 2-41; 3-44; 5-4
Baudratenauswahl · 2-11
Bedienpanels · 3-76
Bedingungsprimitive · 2-63
Beep · 3-49
Beispielprogramme · 3-68
Benachrichtigungsschleife · 4-9
Betriebsmodus · 1-22
binär · 2-57; 2-58
Binärdatei · 5-13
Binärdateien · 5-12
Binärformat · 5-14
Binärmodus · 2-64
Botschaftsattribut · 2-15; 2-20; 3-17; 3-38
Botschaftsdarstellung · 4-3
Botschaftsdefinitionen · 4-7
Botschaftsdeklaration · 3-6
Botschafts-Explorer · 2-19; 3-24
Botschaftsrate · 2-5; 2-39; 5-11
Botschaftsvariable · 3-9
Break · 2-60; 2-61
Break-Bedingung · 2-60
Breakpoint · 2-60; 2-61
BTL-Zyklen · 2-11
BTR-Programmierung · 2-11; 3-46
Busbelastung · 2-23; 4-2; 4-5
Buslast · 2-4; 2-5; 2-10; 2-22; 2-23; 2-36; 2-37; 2-41; 4-2; 4-3
Buslastmessung · 2-41
BUSOFF · 3-30; 3-44; 5-9; 5-12
Busparameter · 2-10; 2-11; 3-68; 5-1
Busstatistik · 2-5; 2-14; 2-41; 2-59; 5-10
Busstatistikanzeige · 4-3
Busstatistikinformationen · 2-41; 2-58
Busstörungen · 1-1
Bus-Timing-Register · 2-12
Bustreiberbox · 5-5
Button · 1-21

C

callAllOnEnvVar() · 3-57

CAN.INI · 2-1; 2-9; 2-22; 2-38; 2-42; 2-58; 2-61; 3-2; 3-41; 4-4; 5-3; 5-5; 5-7; 5-8
CAN-AC2 · 5-2
CANcard · 2-10; 5-2; 5-5
CANcardX · 2-10; 3-14
cancelTimer() · 3-13; 3-19; 3-53
CANdb · 1-1; 1-23; 2-18; 2-19; 2-29; 3-24; 3-74
canOffline() · 3-53
canOnline() · 3-53
CAPL · 2-55
CAPL Sprachbeschreibung · 3-26
CAPL.INI · 3-60
CAPL_DLL_INFO_LIST · 3-42
CAPL-Programm · 2-20; 2-42; 2-43; 3-2; 3-10; 3-26; 3-27; 3-73; 3-74
Checkbox · 1-21
Compiler · 2-10; 3-2; 3-5; 3-15; 3-21; 3-22; 3-23; 3-24; 3-25; 3-36; 3-41; 3-48; 3-50
Compilierzeit · 3-22
Complierzeit · 3-24
ConfigurationRequest · 4-9; 4-10
cos() · 3-20
Cosinus · 3-59
CPC/PP · 3-46; 3-47
CyclicUpdateTime · 4-4

D

Darstellungsformate · 2-9; 2-10
Darstellungsparameter · 2-24; 2-28
Data-Length-Code · 2-20; 3-37
Dateiformat · 3-23; 5-13
Dateinamen · 5-2
Dateitypen · 4-12; 5-2
Dateizugriff · 3-63
Datenbasis · 3-76
Datenbasisattribut · 3-17
Datenbasisauswahldialog · 4-6; 4-7; 4-8
Datenbasiseditor · 3-17
Datendurchsatz · 2-41; 5-8
Datenfluß · 1-4; 2-3; 2-5; 2-6; 2-15; 2-42; 3-3; 3-10; 3-68; 4-2
Datenflußdiagramm · 1-3; 2-4; 2-5; 2-6; 2-7; 2-15; 2-42; 2-59; 4-1; 5-11
Datenquelle · 2-5; 2-10; 2-41; 2-52; 2-58; 2-60; 2-62; 5-6
Datenreduktion · 2-44; 2-60; 4-5
Datenverlust · 2-52; 4-2; 4-3; 4-4
DDE · 4-8; 4-9; 4-10; 4-11; 4-12
Debugging · 3-67; 3-68
Demo-Version · 3-47; 5-1; 5-5
Dezimaldarstellung · 3-48
Dialog · 1-20
Differenzleiste · 2-33
Differenzmarken · 2-33
Differenzmeßmodus · 2-33
DIR · 3-7; 3-8; 3-29; 3-35; 3-37; 3-38; 3-71; 3-72
DLC · 1-3; 2-20; 2-48; 3-7; 3-29; 3-35; 3-37; 3-70; 3-71; 3-72; 5-9; 5-10
double · 3-4; 3-5; 3-27; 3-36; 3-40; 3-42; 3-58; 3-59
Drag and Drop · 3-23
Dropdown-Liste · 1-21
Durchlaßfilter · 2-45
Dynamic · 4-8
DynamicStateRequest · 4-9; 4-10

E

Echtzeitbibliothek · 3-6; 3-41; 4-2; 4-3; 4-4; 4-5; 5-6
Echtzeitfähigkeit · 3-60
Echtzeitmodul · 4-2
Echtzeituhr · 5-7; 5-8
Echtzeitzweig · 4-5
Eingabefeld · 1-21
Einsteckkarte · 2-6; 2-8; 2-10; 2-12; 2-23; 2-62; 4-1; 5-4; 5-7; 5-8
Einzelschrittbetrieb · 2-61; 2-62
Einzelsignalmodus · 2-28; 2-32; 2-33; 2-34
elCount() · 3-5; 3-20; 3-41
Empfangsbotschaften · 2-15
Empfangschip · 3-9
Empfangsfehlerzähler · 3-37
Empfangspuffer · 5-11
Empfangszeitpunkt · 2-10
enablePortBits() · 5-7
Endlosschleife · 2-16
Ereignisprozedur · 3-8
ERROR · 2-41; 5-3; 5-9
ERRORACTIVE · 3-30
Error-Frame · 2-4; 2-20; 2-41; 2-45; 2-46; 2-56; 3-8
ERRORPASSIVE · 3-8; 5-9
Event-Prozeduren · 3-11; 3-26; 3-30; 3-32; 3-39
Exponentialfunktion · 3-59
Extended Identifier · 2-40
Extended-Identifier · 2-20; 3-51

F

Farben im Grafikfenster · 2-36
Fehlerbehebung · 5-3
Fehlercode · 5-9
Fehlerzähler · 3-14; 3-39
Feldgrößenbestimmung · 3-20
Fensteranordnung · 2-42
Fensterbedienung · 2-7
Fenster-Layout · 2-42
fileName() · 3-68
fileReadInt() · 3-62
Filter · 1-4; 2-5; 2-6; 2-15; 2-19; 2-45
Firmware · 5-2; 5-8; 5-9; 5-10; 5-11; 5-12
Fit · 2-40
Font · 2-42
Formatangabe · 3-21; 3-48
Formatbeschreibungsdatei · 5-13
Formate von Konfigurationen · 2-9
Füllstandsanzeige · 4-3
Funktionsblock · 2-7
Funktionsgenerator · 2-50
Funktionstabelle · 3-41

G

Ganzzahlkonstante · 3-29
Ganzzahlvariable · 3-20; 3-21
Ganzzahlwert · 3-17

Ganzzahlwerte · 3-17
Gateway · 3-1; 3-2; 3-9; 3-72
Generatorblock · 1-4; 2-5; 2-6; 2-7; 2-15; 2-43; 2-46; 2-47; 2-48; 2-50; 2-51; 3-3; 5-2
getCardType() · 3-47
getLocalTime() · 3-55
getLocalTimeString() · 3-56
getValue() · 3-12; 3-13; 3-20; 3-56; 3-76; 3-77
Gleitkommakonstante · 3-30
Gleitkommazahlen · 3-4; 3-5; 3-27; 3-36; 3-62
Globale Einstellungen · 2-9
globale Variablen · 3-23
goto · 3-32
Grafikdiagramm · 2-38
Grammatik · 3-26
Grundbild · 2-40
Grundeinstellungen · 2-35

H

Hardwareeinstellungen · 2-1; 5-7
Hauptspeicher · 2-56; 5-5; 5-6; 5-7
hexadezimal · 2-3; 2-13; 2-20; 2-24; 2-64; 3-16; 3-62; 5-2; 5-14
Hexadezimaldarstellung · 2-58; 3-48
Hilfe · 1-23; 2-13; 3-31
Hochlastbetrieb · 4-2
Hot-Spots · 1-4; 2-5; 2-42

I

IEEE · 3-5; 3-27; 3-36
INI-Datei · 2-1; 2-9; 2-22; 2-38; 2-42; 2-58; 2-61; 3-2; 3-41; 4-4; 5-5; 5-7; 5-8
Initialisierung von Umgebungsvariablen · 3-58
Initialisierungsdateien · 5-2
inport() · 3-20; 3-45
Intel · 2-18; 3-58

K

Karteninitialisierung · 5-11
Kartentreiber · 2-14; 2-15; 2-16; 5-1; 5-10; 5-11
Kartenzeitgeber · 2-16
Knoten · 3-76
Kommentar · 2-43; 3-25; 3-26; 3-51; 5-13
Kommentarfeld · 1-21
Konfigurationsdatei · 2-1; 2-2; 2-4; 2-8; 2-9; 5-2; 5-3
Konfigurationsdateiformate · 2-9
Konfigurationsmöglichkeiten · 2-6; 2-36
Konsistenzüberprüfung · 4-8
Kontextmenüs · 1-20
Kontrollkästchen · 1-21
Kopieren von Blöcken · 2-6

L

Langzeitmessung · 2-56
Lastbetrieb · 2-37; 4-3; 4-4; 5-6
Lastsituation · 4-3
Lastübergang · 4-4; 5-6
Laufzeitbibliothek · 5-7
Laufzeitfehler · 3-5; 3-24; 3-41; 3-43
Layout · 2-42
Linientyp · 2-31
Logging-Dateiname · 3-51
Logging-Triggerung · 3-50
Lowspeed-Treiber · 5-7

M

Mausinterrupt · 5-10
Mehrdeutigkeiten · 2-25; 3-19; 3-25; 3-28; 4-6; 4-7; 4-8
Menübedienung · 1-20
Message-Explorer · 2-19
Meßleiste · 2-33
Meßmarke · 2-33
Meßpunkt · 2-31; 2-33; 2-34
Messungsende · 2-3; 2-28; 2-30; 2-32; 2-37; 2-55; 3-8; 3-14
Messungskontrolle · 3-19
Messungsstart · 2-7; 2-10; 2-32; 2-55; 2-59; 3-5; 3-8; 3-20; 3-68; 3-69; 5-5
Meßwerterfassung · 2-31
Mittelungszeit · 2-38; 2-39; 2-41; 4-5
Moduldefinitionsdatei · 3-41
modulo · 3-33
modusabhängiges Signal · 2-49
Modussignal · 2-49
Motorola · 2-18; 3-15; 3-36; 3-58
msTimer · 3-4; 3-5; 3-6; 3-13; 3-27; 3-53; 3-54; 3-69; 3-75
Multiplexbotschaft · 2-49
Multisignalmodus · 2-28; 2-32; 2-33; 2-34; 2-35

N

Nachlaufzeit · 2-8; 2-54; 2-55; 2-56
Netzknoten · 3-76
Nonterminal · 3-26
Normal-Modus · 2-54

O

OCR · 3-45; 3-46
Offline-Modus · 2-21; 2-59; 2-61; 2-62
on key · 3-71
on message · 3-72; 3-73
Online-Modus · 2-5; 2-7; 2-10; 2-14; 2-23; 2-52; 2-62; 4-10; 4-11
Optimierung · 2-27; 2-36; 2-37
Optionsschaltfläche · 1-21
outport() · 3-20; 3-45
output() · 2-44; 3-1; 3-19; 3-43; 3-69; 3-70; 3-71; 3-72; 3-73
output(this) · 2-44; 3-10; 3-11; 3-12; 3-15; 3-36
overrun · 5-5

P

Panes · 3-20; 3-23; 3-24; 3-25
Parameterprüfung · 3-19
PARBROW.INI · 3-23
Peakdarstellung · 2-27
Performance · 2-27; 2-41
PG · 3-27; 3-33; 3-37; 3-40
PGN · 3-35
Piggy-Backs · 5-7
Poke · 4-9
Pollingmodus · 5-7; 5-8
preStart · 3-8; 3-13
Primitive · 2-63
Prioritätsreihenfolge · 3-10
Programmaufbau · 3-26
Programmblock · 2-15; 3-1
Programmstart · 1-3; 2-1; 3-32
Projektdatei · 2-9
Projektdatenbasis · 3-15; 3-36; 3-38; 3-74; 5-2
Projektverzeichnis · 2-8
Prozedurliste · 3-23
Prozedurrumpf · 3-20; 3-21
Punktmeßmodus · 2-33
putValue() · 3-12; 3-20; 3-57; 3-76; 3-77

R

Radiobutton · 1-21
Raster · 2-35
Reaktionszeiten · 4-2; 4-5
Rechengenauigkeit · 3-27
rechte Maustaste · 1-20
Refresh · 2-37; 2-38; 4-5
Rekursion · 3-10; 3-12
Replay-Block · 2-46
Reset · 2-61; 3-44; 5-11
resetCan() · 3-44; 3-45
Ringpuffer · 2-56; 4-2; 4-3; 4-4; 5-6
RTC · 5-7; 5-8
RTR · 3-37; 3-38
runError() · 3-5; 3-24; 3-41; 3-43
Rx-Attribut · 2-15; 2-39
Rx-Buffer-Overrun · 2-41
RXREMOTE · 3-38

S

Selektoren · 3-7; 3-27; 3-36; 3-37
Semantik · 3-26; 3-32
Sendeblock · 2-14
Sendefehlerzähler · 3-37
Sendeprotokolle · 3-78
Sendezweig · 2-14
seqFileClose() · 3-63
seqFileGetBlock() · 3-63
seqFileGetLine() · 3-63
seqFileGetLineSZ() · 3-63
seqFileLoad() · 3-63
seqFileRewind() · 3-64
sequentieller Dateizugriff · 3-63
setLogFileName() · 3-51
setMsgTime() · 3-54
setPortBits() · 5-5; 5-7
setTimer() · 3-5; 3-13; 3-19; 3-53; 3-69; 3-70; 3-71; 5-8
Signal · 2-18; 2-64
Signalauswahldialog · 2-28; 2-31
Signalbezeichner · 2-25; 2-29
Signal-Explorer · 1-13; 1-14; 2-19; 3-24
Signalkurve · 2-31
Signalverlaufsanzeige · 2-37
Signalverlaufsgenerator · 2-50; 2-51
Signalwert · 2-49
Simulationsaufbau · 3-7; 3-8; 3-10; 3-12; 3-23; 3-24; 3-25; 3-41; 3-68; 3-73
sin() · 3-20
Sinus · 3-58; 3-59
SJA 1000 · 2-10; 2-12
Skalierung · 2-40
Soundkarte · 3-49
Speicherschutzverletzungen · 3-6
Sperrfilter · 2-43; 2-45; 4-5
Sprachelemente · 3-26
sprintf() · 2-26; 3-66
Stackoverflow · 3-24; 3-43
Standard Layout · 2-42
Standardabweichung · 2-40; 2-41
Start · 2-61
Statistik · 2-38
Statistikblock · 2-38
Statistikfenster · 2-39
Statistikprotokoll · 2-39; 2-40; 2-41; 4-2; 4-5; 5-11
Step · 2-61
Stop · 2-55
stop() · 2-8; 2-55; 2-61; 3-19; 3-52; 3-53
Suchbedingung · 2-62
Suchen · 2-23
Suchreihenfolge · 4-8
swapLong() · 3-58
swapWord() · 3-58
Symbolauswahldialog · 3-25
Symboldatenbasis · 3-74
Symbolische Datenbasen · 2-17
Symbolische Triggerung · 2-64
Syntax · 3-26; 3-32; 3-33
sysExit() · 3-49
sysMinimize() · 3-49
Systembelastung · 2-27; 2-38; 5-6
Systemfehler · 5-4
Systemmeldungen · 2-41
Systemverzeichnis · 2-1
Systemzeit · 3-54; 3-55; 3-56

T

Tastaturbehandlung · 3-12
Tastaturereignisse · 3-11
Tastencodes · 3-12; 3-38
Terminal · 3-26
Texteditor · 3-23
Thema System · 4-9
Themennamen · 4-8; 4-9
this · 3-12; 3-14; 3-15; 3-36; 3-39
timeDiff() · 3-55

timeNow() · 3-19; 3-54
Timeout · 5-3; 5-5; 5-11
Timer · 3-1; 3-4; 3-27; 3-31; 3-69; 5-6; 5-8
Timerrate · 5-3
Toggle-Trigger-Modus · 2-54
ToOffline · 4-11
ToOfflineCopy · 4-11
ToOnline · 4-11
ToOnlineCopy · 4-11
Topic System · 4-9
Transaktion Execute · 4-10
Trapezfunktion · 2-50
Treiberoptionen · 2-14; 2-58
Trigger · 2-52; 2-54; 2-55; 2-56; 2-58; 2-59; 2-62; 3-2
trigger() · 2-56; 3-2; 3-19; 3-50; 3-74
Triggerbedingung · 2-48; 2-52; 2-56; 2-62; 3-74
Triggerblock · 2-59
Triggermodus · 2-53
Triggertyp · 2-55; 3-74
Triggerung · 2-55; 3-2; 3-50; 3-73; 3-74
Triggerzeitpunkt · 2-21
Trigonometrie · 3-58; 3-59
Tx-Attribut · 2-16; 2-39
TXDATA · 3-35; 3-38
TXREMOTE · 3-35; 3-38
TXREQUEST · 2-14; 3-7; 3-35; 3-37; 3-38
TXREQUESTDATA · 3-35; 3-38
TXREQUESTREMOTE · 3-35; 3-38
TxRq-Attribut · 2-16

U

Überlast · 4-3
Überlastsituation · 4-3; 4-4
Überlastsymbol · 2-59
Überlauf · 5-5; 5-9; 5-11
Übertragungszeit · 2-17
Umgebungsvariable · 3-56; 3-57
Umgebungsvariablen · 3-1; 3-12; 3-13; 3-15; 3-76; 3-77; 5-6
Umrechnungsformel · 2-18; 2-25; 2-26; 2-29; 3-16; 3-36
Unterbrechung · 2-52; 2-60

V

VAN · 3-46; 3-47
Vergrößerungsmodus · 2-6
Verschieben von Blöcken · 2-6
Verzögerungszeiten · 2-14
Verzweigungen · 1-3; 2-4
Verzweigungspunkt · 2-52
Vorlaufzeit · 2-54; 2-55; 2-56
Vorteiler · 2-11
Vorzeichenbehandlung · 2-18

W

Warning Limit · 3-8; 3-14
Werkzeugleiste · 2-30; 3-21; 3-22; 3-23
Wertebereichsanzeige · 2-35
Wertebereichsüberprüfung · 3-5
Wertetabelle · 2-49
Wildcard · 3-28; 3-31
WIN.INI · 3-48
Windows-Timer · 5-6
write() · 2-4; 2-42; 3-14; 3-19; 3-48; 3-66; 3-72
writeToLog() · 3-19; 3-50
writeToLogEx() · 3-19; 3-50

Z

Zahldarstellung · 1-21
Zahlenbasis · 2-3; 2-9; 3-66; 5-12
Zeitauflösung · 5-3
Zeitbasis · 2-62; 5-7
Zeitfenster · 2-54; 2-56
Zeitlupe · 2-23; 2-61
Zeitstempel · 2-16; 2-22; 2-23; 5-1; 5-3
Zeitverwaltung · 3-19; 3-53; 5-8
Zoom · 2-40
Zufallszahlengenerator · 3-60
Zustandsänderung · 5-3
Zwischenablage · 2-42; 3-23
zyklischer Update · 2-22; 2-57; 4-4; 4-5
Zykluszeit · 2-27; 2-38; 3-16; 3-17; 4-4